Abstract of "Toward Solving Penn Treebank Parsing" by Do Kook Choe, Ph.D., Brown University, April 2017.

A natural language parser recovers the latent grammatical structures of sentences. In many natural language processing (NLP) applications, parsing is applied to sentences first and the parses along with their sentences are fed to following NLP systems. For example, Google parses the entire web and applies a series of NLP programs to index the web and the quality of search results depends on the quality of parses. Parsing is difficult because sentences are ambiguous: a sentence has different syntactic structures depending on its meaning. For example, a sentence "Eugene wears a bow tie with polka dots" can have very different meanings depending on what "with polka dots" modifies. It is natural for us humans to infer that "with polka dots" modifies "a bow tie" because we have common sense that "with polka dots" rarely (if not never) describes an action "wears." Computers, however, lack common sense and learn such a relationship from large amounts of texts by just looking for statistical patterns. We explore four ways of improving parsing in this thesis: creating a training data of high quality parses using paraphrases; a model combination technique applied to n-best parsing; a generative reranker based on a language model; a discriminative parser inspired by neural machine translation. Our parse-reranker achieves human-level performance on the standard Penn Treebank dataset.

Toward Solving Penn Treebank Parsing

by
Do Kook Choe
B. A., New York University, 2012
Sc. M., Brown University, 2014

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island
April 2017

© Copyright 2017 by Do Kook Choe

This dissertation by Do Kook Choe is accepted in its present form by
the Department of Computer Science as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date _____ _____
Eugene Charniak, Director

Recommended to the Graduate Council

Date _____ _____
Michael L. Littman, Reader

Date _____ _____
Erik B. Sudderth, Reader
University of California, Irvine

Date _____ _____
Stefanie Tellex, Reader

Approved by the Graduate Council

Date _____ _____
Andrew G. Campbell
Dean of the Graduate School

Acknowledgements

I'd like to thank Eugene Charniak who let me research whatever subjects I wanted to work on. He patiently listened to me when I rambled and welcomed me anytime I bugged him for questions. It was fun working with him and I will miss it. I also like to thank my committee members, Michael Littman, Erik Sudderth and Stefanie Tellex, who guided me through my PhD years at Brown.

I thank Rebecca Mason, Ben Swanson, Chris Tanner, and Byron Wallace for their helpful comments and ideas for my research especially during my first year. I am grateful to David Eisenstat and Dae Il Kim for their help for my first paper at Brown. Thanks to all of them I got my first paper relatively easily and the rest of my PhD years went smoothly.

Additionally thanks to Mohit Bansal, Dave Buchanan, Shay Cohen, Micha Elsner, Yoav Goldberg, Will Headden, Karen Ingraffea, Shashi Narayan, Siddharth Patwardhan, Kapil Thadani for their discussions for my papers.

I also like to thank members of machine learning reading group for their helpful discussions on broad topics of machine learning: Soumya Ghosh, Nakul Gopalan, Michael Hughes, Geng Ji, Jason Pacheco, Zhile Ren

I had great opportunities to work at large industry research labs, Google and IBM, and for that I am grateful to Jennifer Chu-Carroll, Yun-hsuan, David McClosky and Brian Strope.

I thank Mark Steedman and University of Edinburgh for having me visit Edinburgh when Eugene went there for his sabbatical. While I was there I changed my research direction and as a result I got wonderful results which became the two chapters of this thesis.

I am indebted to Eugene Charniak, Michael Collins, Mark Johnson, David McClosky, Mike Steedman and Google for resolving my immigration issue. Thanks to them, I was stress-free in my last PhD year.

I thank my friends who put up with me whining: Yeounoh Chung, Taek-min Jung, Mooryong Kim, Evgenios Kornaropoulos, Jun Ki Lee, Alexandra Papoutsaki.

Finally I'd like thank my mom and dad for their endless support throughout my entire life. Also I thank my sister and brother, 초록 and 찬국, for not hating me after all these years.

엄빠 고맙습니다.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Parsing is a fundamental process humans subconsciously do to understand what they hear and read. For example, when an English speaker hears "Eugene wears a bow tie with polka dots," he or she immediately understands: "Eugene" is a person; "a bow tie" is a piece of fabric "Eugene" puts on around his neck; and "polka dots" are drawn on "a bow tie." Because we humans parse sentences all the time very easily, one may wonder why parsing is challenging for computers . In order for a program to produce the correct parse tree for a sentence, it has to infer many things: "Eugene" is a subject; "wears" is a verb; "a bow tie" is a multi-word object; and "with polka dots" is a phrase that modifies "a bow tie." If it thinks "tie" as a verb during its inference, it will produce a nonsensical parse tree. A parser needs to infer the roles of words and guess what relations these words form. For a sentence of 20 words, a parser needs to make at least 40 decisions and if it guesses them any incorrectly it will produce an incorrect parse tree. Because the parse tree search space is exponential in terms of the number of words in a sentence, developing an advanced inference algorithm is hard.

In order to create machines people can talk to, building human level parsing models is necessary because many natural language processing (NLP) systems read in parsed sentences as their inputs and run their own components to understand what users want. For instance, personal digital assistants such as Google Now and Apple Siri parse transcribed human speeches into syntactic trees which in turn go through subsequent NLP applications that analyze users' requests. When parsers make mistakes, the assistants are not likely to execute what users have intended. A an another example, IBM Watson who won Jeopardy! against the best human players first parses questions and feeds the parses to its other NLP components. Watson's chance of getting answers right heavily depends on how well its parser parses questions. When the parser fails, Watson has a very low probability of producing the correct answers. Many programs in academia as well as in industry run parsing at the beginning of their pipelines.

We briefly review various techniques for building parsing models (Chapter 2) and existing neural network parsing models (Chapter 2) and describe our published contributions to these techniques (Chapter 3, 4, 5). In Chapter 6, we present a greedy parser built on recurrent neural networks.

Chapter 2

Background

2.1 Parsing

Parsing is a task of finding a syntactic tree of a sentence. Given a sentence ($\boldsymbol{x}$), a generative parser produces its grammar ($\boldsymbol{y}$) that satisfies

$$\underset{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})}{\operatorname{argmax}} P(\boldsymbol{x}, \boldsymbol{y}') \tag{2.1}$$

where $\mathcal{Y}(\boldsymbol{x})$ is a set of all possible grammatical structures of $\boldsymbol{x}$ and $P(\boldsymbol{x}, \boldsymbol{y}')$ is the probability of a parse $(\boldsymbol{x}, \boldsymbol{y}')$.[1] Most parsing research goes into learning $P(\boldsymbol{x}, \boldsymbol{y})$ and broadly there are two kinds of parsers depending on how they break down $P(\boldsymbol{x}, \boldsymbol{y})$. A tree-based parser multiplies production rule probabilities to compute the tree probability:

$$P(\boldsymbol{x}, \boldsymbol{y}) = \prod_{r \in \mathcal{R}(\boldsymbol{x}, \boldsymbol{y})} P(r)$$

where $\mathcal{R}(\boldsymbol{x}, \boldsymbol{y})$ is a set of all production rules in tree $(\boldsymbol{x}, \boldsymbol{y})$. On the other hand, a sequence-based one multiplies conditional probabilities of symbols from left to right:

$$P(\boldsymbol{x}, \boldsymbol{y}) = P(\boldsymbol{z}) = \prod_{i=1}^{m} P(z_i | z_1, \cdots, z_{i-1}) \tag{2.2}$$

where $\boldsymbol{z}$ is a sequential form of $(\boldsymbol{x}, \boldsymbol{y})$ [103]. Tree-based parsers tend to be slow because they make global inference whereas sequence-based ones are fast because they make greedy decisions. However a parser breaks down $P(\boldsymbol{x}, \boldsymbol{y})$, it learns to maximize the likelihood of the training data during training and finds the most-likely tree according to Equation (2.1) for a sentence during testing. In Chapter 5, we present a generative model that assign probabilities to trees according to Equation (2.2). As in any machine learning task, a parsing model becomes more accurate as it sees more data. But the

[1] Discriminative parsers model the conditional probability, $P(\boldsymbol{y}' | \boldsymbol{x})$.

problem is that annotating sentences with trees is very time-consuming and expensive and as a result only small gold treebanks are available for training supervised parsers. In an effort to overcome this problem, we explore an alternative method for annotating sentences with trees using paraphrases in Chapter 3. We now look at a simple way of collecting many auto-parsed trees.

2.2 Semi-supervision

Gold treebanks, collections of parses annotated by humans, are small compared to other domains' datasets. For example, all treebanks consist of fewer than 100,000 trees and most treebanks fewer than 10,000 whereas the ImageNet contains millions of labeled images [26]. The small datasets are often a major factor preventing researchers from developing accurate parsing models. To overcome data scarcity, researchers have proposed to parse a large amount of unannotated sentences with existing parsers and trained improved models on this "silver" trees in addition to gold trees [68, 103]. This method is simple and yet very effective. In our work, we also parse millions of trees, on which we train our models, as the "silver" trees guarantee improved performance.

2.3 Reranking

Parsers make mistakes no matter what inferences they use to search through the parse space because they approximate the true tree probabilities. A tree-based model assumes conditional independences to make its inference tractable, which leads to approximation, and a sequence-based one makes a series of local decisions, which obviously is not exact. A parse-reranker, which assigns probabilities to parses, suffers less from the search problem as it can look at the whole tree as opposed to parts of the tree. Discriminative rerankers built on linear classifiers assign probabilities to trees as follows:

$$P(\boldsymbol{y}|\mathcal{Y}'(\boldsymbol{x})) = \frac{\exp(\theta \cdot f(\boldsymbol{y}))}{\sum_{\boldsymbol{y'} \in \mathcal{Y}'(\boldsymbol{x})} \exp(\theta \cdot f(\boldsymbol{y'}))}$$

where θ is a set of model parameters, $f(\boldsymbol{y})$ is a set of features of $\boldsymbol{y}$ and $\mathcal{Y}'(\boldsymbol{x})$ is n trees provided by a parser. The discriminative rerankers have proven to be very effective and been widely used [22, 15], and recently a generative reranker, which assign probabilities to trees according to Equation (2.2), based on LSTMs is proposed and shown to perform significantly better than the discriminative rerankers [31]. In Chapter 5, we demonstrate a simpler generative reranker based on an LSTM language model performs even better and achieves the state of the art performance. In Chapter 6, we present a discriminative parser inspired by the generative reranker.

2.4 Model Combination

For any task, one of the most effective methods of improving existing models is to combine their decision powers. Different techniques for combining parsing models have achieved state of the art

	# sents	# words	sections
train	39382	950028	2-21
dev	1346	32853	24
test	2416	60548	23

Table 2.1: The Penn Treebank statistics.

performance in parsing, and we take a look at two methods that are most relevant to this thesis.

A simple model combination method is similar to majority voting for binary classification. If we are given three binary classifiers, an obvious way of combining their outputs is to have a majority vote. Researchers have applied this idea to parsing as well with some modifications as parsing is a structured prediction task [91]. A naive way of combing three parses of a sentence is to create a new tree with nodes that occur in at least two of the three. In our work, we show that this combination technique can be applied to a single n-best parser (Chapter 4.2).

A product of experts, often called an ensemble, multiplies the probabilities of a tree provided by several parsers [83]. Given n models, it finds the most-likely structure ($\boldsymbol{y}$) satisfying

$$\operatorname*{argmax}_{\boldsymbol{y}' \in \mathcal{Y}} P(\boldsymbol{y}') \propto \operatorname*{argmax}_{\boldsymbol{y}' \in \mathcal{Y}} \prod_{i=1}^{n} P_i(\boldsymbol{y}')$$

where $P_i(\boldsymbol{y}')$ is the probability of $\boldsymbol{y}'$ computed by ith model. In high level, the multiplication weeds out parses that are assigned low probabilities by at least one parser and outputs a tree that all parsers are confident of. An ensemble of models is easy to use and provides nice improvement over an individual model. An ensemble of rerankers performs better than a single reranker (Chapter 5).

2.5 Evaluation

We evaluate parsing models primarily on the Penn Treebank (PTB), which is the most widely used parsing benchmark, and compare our models to existing models in the literature [67]. (See the PTB statistics in Table 2.1.) Given parsed and gold trees, we compute two numbers: precision, the number of correct constituents divided by the number of predicted constituents, and recall, the number of correct constituents divided by the number of gold constituents. F_1, a harmonic mean of precision and recall, tells us how accurate a parsing model is. In general, a model with higher F_1 performs better than another with lower F_1 in terms of all parsing aspects. Thus, we are mainly interested in developing models that achieve high F_1 on the PTB.

Chapter 3

Parsing Paraphrases with Joint Inference

3.1 Introduction

Parsing is the task of reconstructing the syntactic structure from surface text. Many natural language processing tasks use parse trees as a basis for deeper analysis.

The most effective sources of supervision for training statistical parsers are treebanks. Unfortunately, treebanks are expensive, time-consuming to create, and not available for most domains. Compounding the problem, the accuracy of statistical parsers degrades as the domain shifts away from the supervised training corpora [39, 5, 69, 98]. Furthermore, for domains requiring subject matter experts, e.g., law and medicine, it may not be feasible to produce large scale treebanks since subject matter experts generally don't have the necessary linguistic background. It is natural to look for resources that are more easily obtained. In this work, we explore using paraphrases. Unlike parse trees, paraphrases can be produced quickly by humans and don't require extensive linguistic training. While paraphrases are not parse trees, a sentence and its paraphrase may have similar syntactic structures for portions where they can be aligned.

We can improve parsers by jointly parsing a sentence with its paraphrase and encouraging certain types of overlaps in their syntactic structures. As a simple example, consider replacing an unknown word in a sentence with a synonym found in the training data. This may help disambiguate the sentence without changing its parse tree. More disruptive forms of paraphrasing (e.g., topicalization) can also be handled by not requiring strict agreement between the parses.

In this thesis, we use paraphrases to improve parsing inference within and across domains. We develop methods using dual-decomposition (where the parses of both sentences from a dependency parser are encouraged to agree, Chapter 3.3.2) and pair-finding (which can be applied to any n-best parser, Chapter 3.3.3). Some paraphrases significantly disrupt syntactic structure. To counter this, we examine relaxing agreement constraints and building classifiers to predict when joint parsing

How did Bob Marley die? What killed Bob Marley?
How fast does a cheetah run? What is a cheetah's top speed?
He came home unexpectedly. He wasn't expected to arrive home like that.
They were far off and looked tiny. From so far away, they looked tiny.
He turned and bent over the body of the Indian. Turning, he bent over the Indian's body.
No need to dramatize. There is no need to dramatize.

Table 3.1: Example paraphrases from our dataset.

won't be beneficial (Chapter 3.3.4). We show that paraphrases can be exploited to improve cross-domain parser inference for two state-of-the-art parsers, especially on domains where they perform poorly.

3.2 Related Work

Many constituency parsers can parse English newswire text with high accuracy [21, 15, 84, 96, 23]. Likewise, dependency parsers have rapidly improved their accuracy on a variety of languages [32, 73, 79, 61, 107, 63]. There are many approaches tackling the problem of improving parsing accuracy both within and across domains, including self-training/uptraining [69, 85], reranking [21, 69], incorporating word clusters [60], model combination [83], automatically weighting training data [70], and using n-gram counts from large corpora [7]. Using paraphrases falls into the semi-supervised category. As we show later, incorporating paraphrases provides complementary benefits to self-training.

3.2.1 Paraphrases

While paraphrases are difficult to define rigorously [9], we only require a loose definition in this work: a pair of phrases that mean approximately the same thing. Paraphrases can be constructed in various ways: replacing words with synonyms, reordering clauses, adding relative clauses, using negation and antonyms, etc. Table 3.1 lists some example paraphrases.

There are a variety of paraphrase resources produced by humans [28] and automatic methods [37]. Recent works have shown that reliable paraphrases can be crowdsourced at low cost [77, 12, 102]. Paraphrases have been shown to help summarization [20], question answering [30, 33], machine translation [13], and semantic parsing [8]. Paraphrases have been applied to syntactic tasks, such as prepositional phrase attachment and noun compounding, where the corpus frequencies of different syntactic constructions (approximated by web searches) are used to help disambiguate [75]. One method for transforming constructions is to use paraphrase templates.

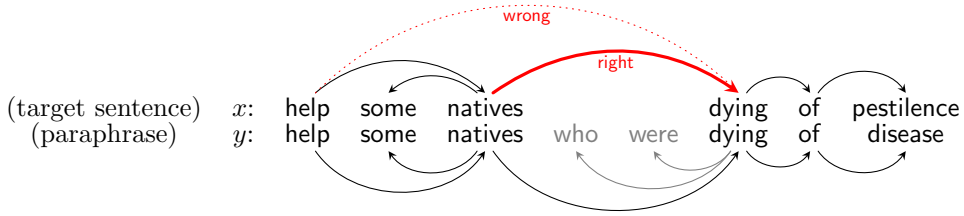


Figure 3.1: An illustration of joint parsing a sentence with its paraphrase. Unaligned words are gray. Joint parsing encourages structural similarity and allows the parser to correct the incorrect arc.

3.2.2 Bilingual Parsing

The closest task to ours is bilingual parsing where sentences and their translations are parsed simultaneously [11]. While our methods differ from those used in bilingual parsing, the general ideas are the same.[1] Translating and paraphrasing are related transformations since both approximately preserve meaning. While syntax is only partially preserved across these transformations, the overlapping portions can be leveraged with joint inference to mutually disambiguate. Existing bilingual parsing methods typically require parallel treebanks for training and parallel text at runtime while our methods only require parallel text at runtime. Since we do not have a parallel paraphrase treebank for training, we cannot directly compare to these methods.

3.3 Jointly Parsing Paraphrases

With a small number of exceptions, parsers typically assume that the parse of each sentence is independent. There are good reasons for this independence assumption: it simplifies parsing inference and oftentimes it is not obvious how to relate multiple sentences (though see [89] for one approach). In this chapter, we present two methods to jointly parse paraphrases without complicating inference steps. Before going into details, we give a high level picture of how jointly parsing paraphrases can help in Figure 3.1. With the baseline parser, the parse tree of the target sentence is incorrect but its paraphrase (parsed by the same parser) is parsed correctly. We use rough alignments to map words across sentence pairs. Note the similar syntactic relations when they are projected across the aligned words.

Our goal is to encourage an appropriate level of agreement between the two parses across alignments. We start by designing "hard" methods which require complete agreement between the parses. However, since parsers are imperfect and alignments approximate, we also develop "soft" methods which allow for disagreements. Additionally, we make procedures to decide whether to use the original (non-joint) parse or the new joint parse for each sentence since joint parses may be worse in cases where the sentences are too different and alignment fails.

[1] Applying our methods to bilingual parsing is left as future work.

3.3.1 Objective

In a typical parsing setting, given a sentence (x) and its paraphrase (y), parsers find $a^*(x)$ and $b^*(y)$ that satisfy the following equation:[2]

$$\begin{aligned} a^*, b^* &= \operatorname*{argmax}_{a \in T(x), b \in T(y)} f(a) + f(b) \\ &= \operatorname*{argmax}_{a \in T(x)} f(a) + \operatorname*{argmax}_{b \in T(y)} f(b) \end{aligned} \tag{3.1}$$

where f is a parse-scoring function and T returns all possible trees for a sentence. f can take many forms, e.g., summing the scores of arcs [32, 73] or multiplying probabilities together [15]. The argmax over a and b of equation (3.1) is separable; parsers make two sentence-level decisions. For joint parsing, we modify the objective so that parsers make one global decision:

$$a^*, b^* = \operatorname*{argmax}_{\substack{a \in T(x), b \in T(y) \\ : c(a,b)=0}} f(a) + f(b) \tag{3.2}$$

where c (defined below) measures the syntactic similarity between the two trees. The smaller $c(a, b)$ is, the more similar a and b are. Intuitively, joint parsers must retrieve the most similar pair of trees with the highest sum of scores.

Constraints

The constraint function, c, ties two trees together using alignments as a proxy for semantic information. An alignment is a pair of words from sentences x and y that approximately mean the same thing. For example, in Figure 3.1, (help^x, help^y) is one alignment and ($\mathsf{pestilence}^x$, $\mathsf{disease}^y$) is another. To simplify joint parsing, we assume the aligned words play the same syntactic roles (which is obviously not always true and should be revisited in future work). c measures the syntactic similarity by computing how many pairs of alignments have different syntactic head relations. For the two trees in Figure 3.1, we see two different relations: ($\mathsf{help} \xrightarrow{x} \mathsf{dying}$, $\mathsf{help} \not\xrightarrow{y} \mathsf{dying}$) and ($\mathsf{natives} \not\xrightarrow{x} \mathsf{dying}$, $\mathsf{natives} \xrightarrow{y} \mathsf{dying}$). The rest have the same relation so $c(a, b) = 2$. As we'll show in Chapter 3.5, the constraints defined above are too restrictive because of this strong assumption. To alleviate the problem, we present ways of appropriately changing constraints later. We now turn to the first method of incorporating constraints into joint parsing.

3.3.2 Constraints via Dual Decomposition

Dual decomposition [88] is well-suited for finding the MAP assignment to equation (3.2). When the parse-scoring function f includes an arc-factored component as in [73], it is straightforward to incorporate constraints as shown in Algorithm 1. Essentially, dual decomposition penalizes relations that are different in two trees by adding/subtracting dual values to/from arc scores. When dual

[2] When it is clear from context, we omit x and y to simplify notation.

Set $u^0(i,j) = 0$ for all $i, j \in E$
for $k = 1$ *to* K **do**
  $a^k = \underset{a \in T(x)}{\text{argmax}} \left(f(a) + \sum_{i,j \in E} u^k(i,j) a(i,j) \right)$
  $b^k = \underset{b \in T(y)}{\text{argmax}} \left(f(b) - \sum_{i,j \in E} u^k(i,j) b(i,j) \right)$
  $v, u^{k+1} = \text{UPDATE}(u^k, \delta^k, a^k, b^k)$
  if $v = 0$ **then return** a^k, b^k
return a^K, b^K

function UPDATE(u, δ, a, b)
  $v = 0, u'(i,j) = 0$ for all $i, j \in E$
  for $i, j \in E$ **do**
    $u'(i,j) = u(i,j) - \delta(a(i,j) - b(i,j))$
    if $a(i,j) \neq b(i,j)$ **then** $v = v + 1$
  return v, u'

Algorithm 1: Dual decomposition for jointly parsing paraphrases pseudocode. E is the set of all possible edges between any pair of aligned words. Given ℓ aligned word pairs, $E = \{1, \ldots, \ell\} \times \{1, \ldots, \ell\}$. $a(i,j)$ is one if the ith aligned word is the head of jth aligned word, zero otherwise. $u(i,j)$ is the dual value of an edge from the ith aligned word to the jth aligned word. δ^k is the step size at kth iteration.

decomposition is applied in Figure 3.1, the arc score of (help $\xrightarrow{x}$ dying) decreases and the score for (natives $\xrightarrow{x}$ dying) increases in the second iteration, which eventually leads the algorithm to favor the latter.

We relax the constraints by employing soft dual decomposition [4] and replacing UPDATE in Algorithm 1 with S-UPDATE from Algorithm 2. The problem with the original constraints is they force every pair of alignments to have the same relation even when some aligned words certainly play different syntactic roles. The introduced slack variable lets some alignments have different relations when parsers prefer them. Penalties bounded by the slack tend to help fix incorrect ones and not change correct parses. In this work, we use a single slack variable but it's possible to have a different slack variable for each type of dependency relation.[3]

3.3.3 Constraints via Pair-finding

One shortcoming of the dual decomposition approach is that it only applies to parse-scoring functions with an arc-factored component. We introduce another method for estimating equation (3.2) that applies to all n-best parsers.

Given the n-best parses of x and the m-best parses of y, Algorithm 3 scans through $n \times m$ pairs of trees and chooses the pair that satisfies equation (3.2). If it finds one pair with $c(a,b) = 0$, then it has found the answer to the equation. Otherwise, it chooses the pair with the smallest $c(a,b)$, breaking

[3] We did pilot experiments with multiple slack variables. Since they showed only small improvements and were harder to tune, we stuck with a single slack variable for remaining experiments.

```
function S-UPDATE(u, δ, a, b, s)
  v = 0, u'(i, j) = 0 for all i, j ∈ E
  for i, j ∈ E do
    t = max(u(i, j) − δ(a(i, j) − b(i, j)), 0)
    u'(i, j) = min(t, s)
    if u'(i, j) ≠ 0, u'(i, j) ≠ s then
      v = v + 1
  return v, u'
```

Algorithm 2: The new UPDATE function of soft dual decomposition for joint parsing. It projects all dual values between 0 and $s \geq 0$. s is a slack variable that allows the algorithm to avoid satisfying some constraints.

```
function PAIR-FINDING(a_{1:n}, b_{1:m})
  Set a, b = null, min = ∞, max = −∞
  for i = 1 to n do
    for j = 1 to m do
      v = C (a_i, b_j)
      sum = f(a_i) + f(b_j)
      if v < min then
        a = a_i, b = b_j
        min = v, max = sum
      else if v = min, sum > max then
        a = a_i, b = b_j
        max = sum
  return a, b

function C(a, b)
  v = 0
  for i, j ∈ E do
    if a(i, j) ≠ b(i, j) then  v = v + 1
  return v
```

Algorithm 3: The pair-finding scheme with a constraint function, c. $a_{1:n}$ are the n-best trees of x and $b_{1:m}$ are the m-best of y.

ties using the scores of the parses ($f(a) + f(b)$). This algorithm is well suited for finding solutions to the equation but the solutions are not necessarily good trees due to overly hard constraints.

The algorithm often finds bad trees far down the n-best list because it is mainly interested in retrieving pairs of trees that satisfy all constraints. Parsers find such pairs with low scores if they are allowed to search through unrestricted space. To mitigate the problem, we shrink the search space by limiting n. Reducing the search space relies on the fact that higher ranking trees are more likely to be correct than the lower ranking ones. Note that we decrease n because we are interested in recovering the tree of the target sentence, x. m should also be decreased to improve the parse of its paraphrase, y.

REL	REL + REL_p
REL + REL_p + REL_{gp}	REL + REL_{gp}
CP	CP + CP_p
CP + CP_p + CP_{gp}	CP + CP_{gp}
REL + CP	REL + CP + CP_p
REL + CP_p + REL_{gp}	

Table 3.2: Feature templates: REL is the dependency relation between the word and its parent. CP is the coarse part-of-speech tag (first two letters) of a word. p and gp select the parent and grandparent of the word respectively.

3.3.4 Logistic Regression

One caveat of the previous two proposed methods is that they do not know whether the original or joint parse of x is more accurate. Sometimes they increase agreement between the parses at the cost of accuracy. To remedy this problem, we use a classifier (specifically logistic regression) to determine whether a modified tree should be used. The classifier can learn the error patterns produced by each method.

Features

Classifier features use many sources of information: the target sentence x and its paraphrase y, the original and new parses of x (a^0 and a), and the alignments between x and y.

Crossing Edges How many arcs cross when alignments are drawn between paraphrases on a plane divided by the length of x. It roughly measures how many reorderings are needed to change x to y.

Non-projective Edges Whether there are more non-projective arcs in new parse (a) than the original (a^0).

Sentence Lengths Whether the length of x is smaller than that of y. This feature exists because baseline parsers tend to perform better on shorter sentences.

Word Overlaps The number of words in common between x and y normalized by the length of x.

Parse Structure Templates The feature generator goes through every word in $\{a^0, a\}$ and sets the appropriate boolean features from Table 3.2. Features are prefixed by whether they come from a^0 or a.

3.4 Data and Programs

This chapter describes our paraphrase dataset, parsers, and other tools used in experiments.

	Development					Test				
	BNC	Brown	QB	WSJ	Total	BNC	Brown	QB	WSJ	Total
Sentences	247	558	843	352	2,000	247	558	844	351	2,000
Tokens	4,297	7,937	8,391	5,924	26,549	4,120	8,025	8,253	5,990	26,388
Tokens$^{\|\|}$	4,372	8,088	8,438	6,122	27,020	4,272	8,281	8,189	6,232	26,974
Word types	1,727	2,239	2,261	1,955	6,161	1,710	2,337	2,320	1,970	6,234
Word types$^{\|\|}$	1,676	2,241	2,261	1,930	6,017	1,675	2,335	2,248	1,969	6,094
OOV	11.2	5.1	5.4	2.4	5.6	11.5	5.1	5.8	2.2	5.7
OOV$^{\|\|}$	8.6	4.7	5.4	2.6	5.1	9.3	4.8	6.0	2.4	5.3
Tokens/sent.	17.4	14.2	10.0	16.8	13.3	16.7	14.4	7.8	17.1	13.2
Avg. aligned	13.1	10.5	6.9	13.0	9.7	12.6	10.7	6.7	13.0	9.7

Table 3.3: Statistics for the four corpora of the paraphrase dataset. Most statistics are counted from sentences with gold trees, including punctuation. $\|\|$ indicates the statistic is from the paraphrased sentences. "Avg. aligned" is the average number of aligned tokens from the original sentences using Meteor. OOV is the percentage of tokens not seen in the WSJ training.

3.4.1 Paraphrase Dataset

To evaluate the efficacy of the proposed methods of jointly parsing paraphrases, we built a corpus of paraphrases where one sentence in a pair of paraphrases has a gold tree.[4] We randomly sampled 4,000 sentences[5] from four gold treebanks: Brown, British National Corpus (BNC), QuestionBank[6] (QB) and Wall Street Journal (section 24) [36, 35, 54, 67]. A linguist provided a paraphrase for each sampled sentence according to these instructions:

> The paraphrases should more or less convey the same information as the original sentence. That is, the two sentences should logically entail each other. The paraphrases should generally use most of the same words (but not necessarily in the same order). Active/passive transforms, changing words with synonyms, and rephrasings of the same idea are all examples of transformations that paraphrases can use (others can be used too). They can be as simple as just changing a single word in some cases (though, ideally, a variety of paraphrasing techniques would be used).

We also provided 10 pairs of sentences as examples. We evaluate our methods only on the sampled sentences from the gold corpora because the new paraphrases do not include syntactic trees. The data was divided into development and testing sets such that development and testing share the same distribution over the four corpora. Paraphrases were tokenized by the BLLIP tokenizer. See Table 3.3 for statistics of the dataset.[7]

[4] The dataset is available upon request.

[5] We use sentences with 6 to 25 tokens to keep the paraphrasing task in the nontrivial to easy range.

[6] With Stanford's updates: `http://nlp.stanford.edu/data/QuestionBank-Stanford.shtml`

[7] The distribution over four corpora is skewed because each corpus has a different number of sentences within length constraints. Samples are collected uniformly over all sentences that satisfy the length criterion.

3.4.2 Meteor Word Aligner

We use Meteor, a monolingual word aligner [27], to find alignments between paraphrases. It uses the exact matches, stems, synonyms, and paraphrases[8] to form these alignments. Because it uses paraphrases, it sometimes aligns multiple words from sentence x to one or more words from sentence y or vice versa. We ignore these multiword alignments because our methods currently only handle single word alignments. In pilot experiments, we also tried using a simple aligner which required exact word matches. Joint parsing with simpler alignments improved parsing accuracy but not as much as Meteor.[9] Thus, all results in Chapter 3.5 use Meteor for word alignment. On average across the four corpora, 73% of the tokens are aligned.

3.4.3 Parsers

We use a dependency and constituency parser for our experiments: RBG and BLLIP. RBG parser [63] is a state-of-the-art dependency parser.[10] It is a third-order discriminative dependency parser with low-rank tensors as part of its features. BLLIP [15] is a state-of-the-art constituency parser, which is composed of a generative parser and a discriminative reranker.[11]

To train RBG and BLLIP, we used the standard WSJ training set (sections 2–21, about 40,000 sentences).[12] We also used the self-trained BLLIP parsing model which is trained on an additional two million Gigaword parses generated by the BLLIP parser [68].

3.4.4 Logistic Regression

We use the logistic regression implementation from Scikit-learn[13] with hand-crafted features from Section 3.3.4. The classifier decides to whether to keep the parse trees from the joint method. When it decides to disregard them, it returns the parse from the baseline parser. We train a separate classifier for each joint method.

3.5 Experiments

We ran all tuning and model design experiments on the development set. For the final evaluation, we tuned parameters on the development set and evaluate them on the test set. Constituency trees were converted to basic non-collapsed dependency trees using Stanford Dependencies [25].[14] We report

[8] Here paraphrase means a single/multiword phrase that is semantically similar to another single/multiword.

[9] The pilot was conducted on fewer than 700 sentence pairs before all paraphrases were created. We give Meteor tokenized paraphrases with capitalization. Maximizing accuracy rather than coverage worked better in pilot experiments.

[10] `http://github.com/taolei87/RBGParser`, 'master' version from June 24th, 2014.

[11] `http://github.com/BLLIP/bllip-parser`

[12] RBG parser requires predicted POS tags. We used the Stanford tagger [101] to tag WSJ and paraphrase datasets. Training data was tagged using 20-fold cross-validation and the paraphrases were tagged by a tagger trained on all of WSJ training.

[13] `http://scikit-learn.org`

[14] Version 1.3.5, previously numbered as version 2.0.5

	Avg	BNC	Brown	QB	WSJ
RBG	86.4	89.2	**90.9**	75.8	93.7
+ Dual	84.7	87.5	87.8	76.0	91.0
+ S-Dual	**86.8**	**89.8**	**90.9**	**76.5**	**94.0**

Table 3.4: Comparison of hard and soft dual decomposition for joint parsing (development section, UAS).

unlabeled attachment scores (UAS) for all experiments and labeled attachment scores (LAS) as well in final evaluation, ignoring punctuation. Averages are micro-averages across all sentences.

3.5.1 Dual Decomposition

Since BLLIP is not arc-factored, these experiments only use RBG. Several parameters need to be fixed beforehand: the slack constant (s), the learning rate (δ), and the maximum number of iterations (K). We set $\delta^0 = 0.1$ and $\delta^k = \frac{\delta^0}{2^t}$ where t is the number of times the dual score has increased [90]. We choose $K = 20$. These numbers were chosen from pilot studies. The slack variable ($s = 0.5$) was tuned with a grid search on values between 0.1 and 1.5 with interval 0.1. We chose a value that generalizes well across four corpora as opposed to a value that does very well on a single corpus. As shown in Table 3.4, joint parsing with hard dual decomposition performs worse than independent parsing (RBG). This is expected because hard dual decomposition forces every pair of alignments to form the same relation even when they should not. With relaxed constraints (S-Dual), joint parsing performs significantly better than independent parsing. Soft dual decomposition improves across all domains except for Brown (where it ties).

3.5.2 Pair-finding

These experiments use the 50-best trees from BLLIP parser. When converting to dependencies, some constituency trees map to the same dependency tree. In this case, trees with lower rankings are dropped. Like joint parsing with hard dual decomposition, joint parsing with unrestricted pair-finding ($n = 50$) allows significantly worse parses to be selected (Table 3.5). With small n values, pair-finding improves over the baseline BLLIP parser.[15] Experiments with self-trained BLLIP exhibit similar results so we use $n = 2$ for all other experiments. Interestingly, each corpus has a different optimal value for n which suggests we might improve accuracy further if we know the domain of each sentence.

3.5.3 Logistic Regression

The classifier is trained on sentences where parse scores (UAS) of the proposed methods are higher or lower than those of the baselines[16] from the development set using leave-one-out cross-validation. We use random greedy search to select specific features from the 15 feature templates defined in

[15] Decreasing m did not lead to further improvement and thus we don't report the results of changing m.
[16] We only use sentences with different scores to limit ceiling effects.

n	Avg	BNC	Brown	QB	WSJ
1	89.5	91.1	91.6	83.3	**94.2**
2	**90.0**	91.4	**92.3**	84.1	94.1
3	89.8	91.5	92.0	**84.2**	93.9
5	89.2	**91.9**	91.4	83.0	93.2
10	87.9	90.5	90.3	81.4	92.2
50	86.3	90.2	88.7	78.6	91.1

Table 3.5: UAS of joint parsing using the pair-finding scheme with various n values on the development portion. $n = 1$ is the baseline BLLIP parser and $n > 1$ is BLLIP with pair-finding.

	Avg	BNC	Brown	QB	WSJ
RBG	86.4	89.2	90.9	75.8	93.7
+ S-Dual	86.8	89.8	90.9	**76.5**	**94.0**
+ Logit	**86.9**	**89.8**	**91.1**	**76.5**	**94.0**
BLLIP	89.5	91.1	91.6	83.3	94.2
+ Pair	90.0	**91.4**	**92.3**	84.1	94.1
+ Logit	**90.3**	91.3	92.1	**85.2**	**94.3**
BLLIP-ST	90.1	92.7	92.3	84.3	93.8
+ Pair	90.7	**93.5**	92.5	85.6	93.8
+ Logit	**91.1**	93.3	**92.6**	**86.7**	**93.9**

Table 3.6: Effect of using logistic regression on top of each method (UAS). Leave-one-out cross-validation is performed on the development data. +X means augmenting the above system with X.

Chapter 3.3.4. Features seen fewer than three times in the development are thrown out. Separate regression models are built for three different parsers. The logistic regression classifier uses an L_1 penalty with regularization parameter $C = 1$.

Logistic regression experiments are reported in Table 3.6. All parsers benefit from employing logistic regression models on top of paraphrase methods. BLLIP experiments show a larger improvement than RBG. This may be because BLLIP cannot use soft constraints so its errors are more pronounced.

3.5.4 Final Evaluation

We evaluate the three parsers on the test set using the tuned parameters and logistic regression models from above. Joint parsing with paraphrases significantly improves accuracy for all systems (Table 3.7). Self-trained BLLIP with logistic regression is the most accurate, though RBG with S-Dual provides the most consistent improvements.

Joint parsing without logistic regression (RBG + S-Dual) is more accurate than independent parsing (RBG) overall. With the help of logistic regression, the methods do at least as well as their baseline counterparts on all domains with the exception of self-trained BLLIP on BNC. We believe that the drop on BNC is largely due to noise as our BNC test set is the smallest of the four. As on development, logistic regression does not change the accuracy much over the RBG parser with soft

	Avg	BNC	Brown	QB	WSJ
RBG	86.7 (81.3)	89.3 (83.7)	90.2 (84.1)	77.0 (71.0)	93.7 (89.9)
+ S-Dual	**87.3 (81.7)**	89.6 (83.8)	**90.7 (84.6)**	**78.1 (71.8)**	**94.0 (90.2)**
+ Logit	87.2 (81.6)	**89.7 (83.9)**	90.6 (84.5)	77.9 (71.7)	93.8 (89.9)
BLLIP	89.6 (86.1)	90.6 (87.2)	91.7 (87.9)	83.6 (79.9)	94.3 (91.6)
+ Pair	90.1 (86.5)	**90.8 (87.3)**	**92.1 (88.4)**	84.7 (80.7)	94.4 (91.6)
+ Logit	**90.3 (86.8)**	90.6 (87.2)	91.9 (88.1)	**85.5 (81.7)**	**94.5 (91.7)**
BLLIP-ST	90.4 (87.0)	**91.8 (88.3)**	92.7 (89.0)	84.8 (81.2)	**94.3 (91.4)**
+ Pair	90.5 (87.1)	91.1 (87.6)	92.7 (89.1)	85.5 (81.8)	94.2 **(91.4)**
+ Logit	**91.0 (87.6)**	91.4 (88.0)	**92.9 (89.3)**	**86.6 (82.9)**	**94.3 (91.4)**

Table 3.7: Final evaluation on testing data. Numbers are unlabeled attachment score (labeled attachment score). +X indicates extending the above system with X. BLLIP-ST is BLLIP using the self-trained model. Coloring indicates a significant difference over baseline ($p < 0.01$).

dual decomposition.

Joint parsing provides the largest gains on QuestionBank, the domain with the lowest baseline accuracies. This fits with our goal of using paraphrases for domain adaptation — parsing with paraphrases helps the most on domains furthest from our training data.

3.5.5 Error analysis

We analyzed the errors from RBG and BLLIP along several dimensions: by dependency label, sentence length, dependency length, alignment status (whether a token was aligned), percentage of tokens aligned in the sentence, and edit distance between the sentence pairs. Most errors are fairly uniformly distributed across these dimensions and indicate general structural improvements when using paraphrases. BLLIP saw a 2.2% improvement for the ROOT relation, though RBG's improvement here was more moderate. For sentence lengths, BLLIP obtains larger boosts for shorter sentences while RBG's are more uniform. RBG gets a 1.4% UAS improvement on longer dependencies (6 or more tokens) while shorter dependencies are more modestly improved by about 0.3-0.5% UAS. Surprisingly, alignment information provides no signal as to whether accuracy improves.

3.6 Conclusions and Future Work

Our methods of incorporating paraphrases improve parsing across multiple domains for state-of-the-art constituency and dependency parsers. We leverage the fact that paraphrases often express the same semantics with similar syntactic realizations. These provide benefits even on top of self-training, another domain adaptation technique.

Since paraphrases are not available at most times, our methods may seem limited. However, there are several possible use cases. The best case scenario is when users can be directly asked to rephrase a question and provide a paraphrase. For instance, question answering systems can ask users to rephrase questions when an answer is marked as wrong by users. Another option is to use crowdsourcing to quickly create a paraphrase corpus [77, 12, 102]. As part of future work, we

plan to integrate existing larger paraphrase resources, such as WikiAnswers [33] and PPDB [37]. WikiAnswers provides rough equivalence classes of questions. PPDB includes phrasal and syntactic alignments which could supplement our existing alignments or be used as proxies for paraphrases. While these resources are noisy, the quantity of data may provide additional robustness. Lastly, integrating our methods with paraphrase detection or generation systems could help provide paraphrases on demand.

There are many other ways to extend this work. Poor alignments are one of the larger sources of errors and improving alignments could help dramatically. One simple extension is to use multiple paraphrases and their alignments instead of just one. More difficult would be to learn the alignments jointly while parsing and adaptively learn how alignments affect syntax. Our constraints can only capture certain types of paraphrase transformations currently and should be extended to understand common tree transformations for paraphrases [46].

Chapter 4

Syntactic Parse Fusion

4.1 Introduction

Researchers have proposed many algorithms to combine parses from multiple parsers into one final parse [47, 106, 91, 80, 34, 83, 51, 50, 72, 94, 76]. These new parses are substantially better than the originals: [108] combine outputs from multiple n-best parsers and achieve an F_1 of 92.6% on the WSJ test set, a 0.5% improvement over their best n-best parser. Model combination approaches tend to fall into the following categories: *hybridization*, where multiple parses are combined into a single parse; *switching*, which picks a single parse according to some criteria (usually a form of voting); *grammar merging* where grammars are combined before or during parsing; and *stacking*, where one parser sends its prediction to another at runtime. All of these have at least one of the caveats that (1) overall computation is increased and runtime is determined by the slowest parser and (2) using multiple parsers increases the system complexity, making it more difficult to deploy in practice. In this thesis, we describe a simple hybridization extension ("fusion") which obtains much of hybridization's benefits while using only a single n-best parser and minimal extra computation. Our method treats each parse in a single parser's n-best list as a parse from n separate parsers. We then adapt parse combination methods [47, 91, 34] to fuse the constituents from the n parses into a single tree. We empirically show that six n-best parsers benefit from parse fusion across six domains, obtaining state-of-the-art results. These improvements are complementary to other techniques such as reranking and self-training. Our best system obtains an F_1 of 92.6% on WSJ section 23, a score previously obtained only by combining the outputs from multiple parsers. A reference implementation is available as part of BLLIP Parser at `http://github.com/BLLIP/bllip-parser/`

4.2 Fusion

Henderson and Brill (1999) [47] propose a method to combine trees from m parsers in three steps: populate a chart with constituents along with the number of times they appear in the trees; remove

any constituent with count less than $m/2$ from the chart; and finally create a final tree with all the remaining constituents. Intuitively their method constructs a tree with constituents from the majority of the trees, which boosts precision significantly. Henderson and Brill [47] show that this process is guaranteed to produce a valid tree. Sagae and Levie (2006) [91] generalize this work by reparsing the chart populated with constituents whose counts are above a certain threshold. By adjusting the threshold on development data, their generalized method balances precision and recall. Fossum and Knight (2009) [34] further extend this line of work by using n-best lists from multiple parsers and combining productions in addition to constituents. Their model assigns sums of joint probabilities of constituents and parsers to constituents. Surprisingly, exploiting n-best trees does not lead to large improvement over combining 1-best trees in their experiments.

Our extension takes the n-best trees from a parser as if they are 1-best parses from n parsers, then follows the reparsing [91]. Parses are weighted by the estimated probabilities from the parser. Given n trees and their weights, the model computes a constituent's weight by summing weights of all trees containing that constituent. Concretely, the weight of a constituent spanning from ith word to jth word with label ℓ is

$$c_\ell(i \to j) = \sum_{k=1}^{n} W(k) C_\ell^k(i \to j) \tag{4.1}$$

where $W(k)$ is the weight of kth tree and $C_\ell^k(i \to j)$ is one if a constituent with label ℓ spanning from i to j is in kth tree, zero otherwise. After populating the chart with constituents and their weights, it throws out constituents with weights below a set threshold t. Using the threshold $t = 0.5$ emulates the method [47] in that it constructs the tree with the constituents in the majority of the trees. The CYK parsing algorithm is applied to the chart to produce the final tree.

Note that populating the chart is linear in the number of words and the chart contains substantially fewer constituents than charts in well-known parsers, making this a fast procedure.

4.2.1 Score distribution over trees

We assume that n-best parsers provide trees along with some kind of scores (often probabilities or log probabilities). Given these scores, a natural way to obtain weights is to normalize the probabilities. However, parsers do not always provide accurate estimates of parse quality. We may obtain better performance from parse fusion by altering this distribution and passing scores through a nonlinear function, $f(\cdot)$. The kth parse is weighted:

$$W(k) = \frac{f(\text{SCORE}(k))}{\sum_{i=1}^{n} f(\text{SCORE}(i))} \tag{4.2}$$

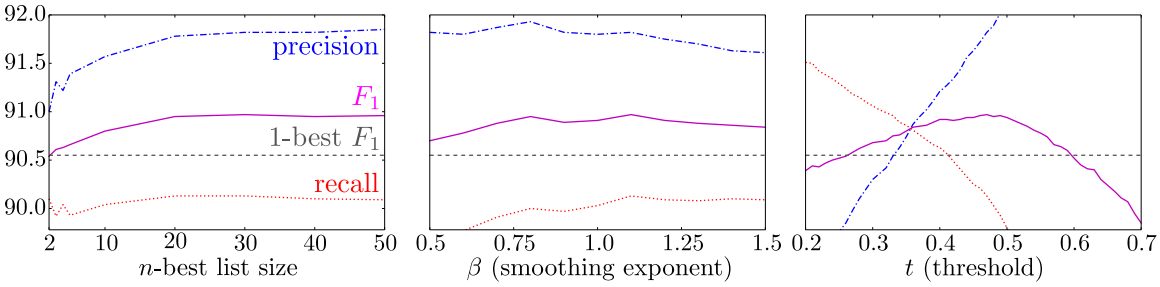


Figure 4.1: Tuning parameters independently for BLLIP and their impact on F_1 for WSJ section 24 (solid purple line). For each graph, non-tuned parameters were set at the optimal configuration for BLLIP ($n = 30$, $\beta = 1.1$, $t = 0.47$). The dashed grey line represents the 1-best baseline at 90.6% F_1.

where SCORE(i) is the score of ith tree.[1] We explore the family of functions $f(x) = x^\beta$ which can smooth or sharpen the score distributions. This includes a tunable parameter, $\beta \in \mathbb{R}_0^+$:

$$W(k) = \frac{\text{SCORE}(k)^\beta}{\sum_{i=1}^{n} \text{SCORE}(i)^\beta} \tag{4.3}$$

Employing $\beta < 1$ flattens the score distribution over n-best trees and helps over-confident parsers. On the other hand, having $\beta > 1$ skews the distribution toward parses with higher scores and helps under-confident parsers. Note that setting $\beta = 0$ weights all parses equally and results in majority voting at the constituent level. We leave developing other nonlinear functions for fusion as future work.

4.3 Experiments

Corpora: Parse fusion is evaluated on British National Corpus (BNC), Brown, GENIA, Question Bank (QB), Switchboard (SB) and Wall Street Journal (WSJ) [35, 36, 55, 54, 40, 67]. WSJ is used to evaluate in-domain parsing, the remaining five are used for out-of-domain. For divisions, we use tune and test splits [5] for Brown, McClosky's test PMIDs[2] for GENIA, Stanford's test splits[3] for QuestionBank, and articles 4000–4153 for Switchboard.

Parsers: The methods are applied to six widely used n-best parsers: Charniak [14], Stanford [58], BLLIP [15], Self-trained BLLIP [68][4], Berkeley [84], and Stanford RNN [96]. The list of parsers and their accuracies on the WSJ test set is reported in Table 4.1. We convert to Stanford Dependencies (basic dependencies, version 3.3.0) and provide dependency metrics (UAS, LAS) as well.

Supervised parsers are trained on the WSJ training set (sections 2–21) and use section 22 or 24 for development. Self-trained BLLIP is self-trained using two million sentences from Gigaword and Stanford RNN uses word embeddings trained from larger corpora.

[1] For parsers that return log probabilities, we turn these into probabilities first.
[2] `http://nlp.stanford.edu/~mcclosky/biomedical.html`
[3] `http://nlp.stanford.edu/data/QuestionBank-Stanford.shtml`
[4] Using the '`WSJ+Gigaword-v2`' BLLIP model.

Parser	F_1	UAS	LAS
Stanford	85.4	90.0	87.3
Stanford RNN	89.6	92.9	90.4
Berkeley	90.0	93.5	91.2
Charniak	89.7	93.2	90.8
BLLIP	91.5	94.4	92.0
Self-trained BLLIP	92.2	94.7	92.2

Table 4.1: Six parsers along with their 1-best F_1 scores, unlabeled attachment scores (UAS) and labeled attachment scores (LAS) on WSJ section 23.

Parser	WSJ	Brown
BLLIP	90.6	85.7
+ Fusion	91.0	86.0
+ Majority voting ($\beta = 0$)	89.1	83.8
+ Rank-based weighting	89.3	84.1

Table 4.2: F_1 of a baseline parser, fusion, and baselines on development sections of corpora (WSJ section 24 and Brown tune).

Parameter tuning: There are three parameters for our fusion process: the size of the n-best list ($2 < n \leq 50$), the smoothing exponent from Chapter 4.2.1 ($\beta \in [0.5, 1.5]$ with 0.1 increments), and the minimum threshold for constituents ($t \in [0.2, 0.7]$ with 0.01 increments). We use grid search to tune these parameters for two separate scenarios. When parsing WSJ (in-domain), we tune parameters on WSJ section 24. For the remaining corpora (out-of-domain), we use the tuning section from Brown. Each parser is tuned separately, resulting in 12 different tuning scenarios. In practice, though, in-domain and out-of-domain tuning regimes tend to pick similar settings within a parser. Across parsers, settings are also fairly similar (n is usually 30 or 40, t is usually between 0.45 and 0.5). While the smoothing exponent varies from 0.5 to 1.3, setting $\beta = 1$ does not significantly hurt accuracy for most parsers.

To study the effects of these parameters, Figure 4.1 shows three slices of the tuning surface for BLLIP parser on WSJ section 24 around the optimal settings ($n = 30, \beta = 1.1, t = 0.47$). In each graph, one of the parameters is varied while the other is held constant. Increasing n-best size improves accuracy until about $n = 30$ where there seems to be sufficient diversity. For BLLIP, the smoothing exponent (β) is best set around 1.0, with accuracy falling off if the value deviates too much. Finally, the threshold parameter is empirically optimized a little below $t = 0.5$ (the value suggested by [47]). Since score values are normalized, this means that constituents need roughly half the "score mass" in order to be included in the chart. Varying the threshold changes the precision/recall balance since a high threshold adds only the most confident constituents to the chart [91].

Baselines: Table 4.2 gives the accuracy of fusion and baselines for BLLIP on the development corpora. Majority voting sets $n = 50, \beta = 0, t = 0.5$ giving all parses equal weight and results in constituent-level majority voting. We explore a rank-based weighting which ignores parse probabilities and weight parses only using the rank: $W_{\text{rank}}(k) = 1/(2^k)$. These show that accurate parse-level

Parser	BNC	Brown	GENIA	SB	QB	WSJ
Stanford	78.4 / 79.6	80.7 / 81.6	73.1 / 73.9	67.0 / 67.9	78.6 / 80.0	85.4 / 86.2
Stanford RNN	82.0 / 82.3	84.0 / 84.3	76.0 / 76.2	70.7 / 71.2	82.9 / 83.6	89.6 / 89.7
Berkeley	82.3 / 82.9	84.6 / 84.6	76.4 / 76.6	74.5 / 75.1	86.5 / 85.9	90.0 / 90.3
Charniak	82.5 / 83.0	83.9 / 84.6	74.8 / 75.7	76.8 / 77.6	85.6 / 86.3	89.7 / 90.1
BLLIP	84.1 / 84.7	85.8 / 86.0	76.7 / 77.1	79.2 / 79.5	88.1 / 88.9	91.5 / 91.7
Self-trained BLLIP	85.2 / 85.8	87.4 / 87.7	77.8 / 78.2	80.9 / 81.7	89.5 / 89.5	92.2 / 92.6
Model combination	86.6	87.7	79.4	80.9	89.3	92.5

Table 4.3: Evaluation of the constituency fusion method on six parsers across six domains. x/y indicates the F_1 from the baseline parser (x) and the baseline parser with fusion (y) respectively. Blue indicates a statistically significant difference between fusion and its baseline parser ($p < 0.01$).

scores are critical for good performance.

Final evaluation: Table 4.3 gives our final results for all parsers across all domains. Results in blue are significant at $p < 0.01$ using a randomized permutation test. Fusion generally improves F_1 for in-domain and out-of-domain parsing by a significant margin. For the self-trained BLLIP parser, in-domain F_1 increases by 0.4% and out-of-domain F_1 increases by 0.4% on average. Berkeley parser obtains the smallest gains from fusion since Berkeley's n-best lists are ordered by factors other than probabilities. As a result, the probabilities from Berkeley can mislead the fusion process.

We also compare against model combination [91] using our reimplementation . For these results, all six parsers were given equal weight. The threshold was set to 0.42 to optimize model combination F_1 on development data (similar to Setting 2 for constituency parsing [91]). Model combination performs better than fusion on BNC and GENIA, but surprisingly fusion outperforms model combination on three of the six domains (not usually not by a significant margin). With further tuning (e.g., specific weights for each constituent-parser pair), the benefits from model combination should increase.

Multilingual evaluation: We evaluate fusion with the Berkeley parser on Arabic [66, 43], French [1], and German [10] from the SPMRL 2014 shared task [93] but did not observe any improvement. We suspect this has to do with the same ranking issues seen in the Berkeley Parser's English results. On the other hand, fusion helps the parser [76] on the German NEGRA treebank [95] to improve from 80.9% to 82.4%.

Runtime: As discussed in Chapter 4.2, fusion's runtime overhead is minimal. Reranking parsers (e.g., BLLIP and Stanford RNN) already need to perform n-best decoding as input for the reranker. Using a somewhat optimized implementation fusion in C++, the overhead over BLLIP parser is less than 1%.

Discussion: Why does fusion help? It is possible that a parser's n-list and its scores act as a weak approximation to the full parse forest. As a result, fusion seems to provide part of the benefits seen in forest reranking [49].

Results [34] imply that fusion and model combination might not be complementary. Both n-best lists and additional parsers provide syntactic diversity. While additional parsers provide greater diversity, n-best lists from common parsers are varied enough to provide improvements for parse

hybridization.

We analyzed how often fusion produces completely novel trees. For BLLIP on WSJ section 24, this only happens about 11% of the time. Fusion picks the 1-best tree 72% of the time. This means that for the remaining 17%, fusion picks an existing parse from the rest of the n-list, acting similar to a reranker. When fusion creates unique trees, they are significantly better than the original 1-best trees (for the 11% subset of WSJ 24, F_1 scores are 85.5% with fusion and 84.1% without, $p < 0.003$). This contrasts with findings [71] where novel predictions from model combination (stacking) were worse than baseline performance. The difference is that novel predictions with fusion better incorporate model confidence whereas when stacking, a novel prediction is less trusted than those produced by one or both of the base parsers.

Preliminary extensions: Here, we summarize two extensions to fusion which have yet to show benefits. The first extension explores applying fusion to dependency parsing. We explored two ways to apply fusion when starting from constituency parses: (1) fuse constituents and then convert them to dependencies and (2) convert to dependencies then fuse the dependencies [91]. Approach (1) does not provide any benefit (LAS drops between 0.5% and 2.4%). This may result from fusion's artifacts including unusual unary chains or nodes with a large number of children — it is possible that adjusting unary handling and the precision/recall tradeoff may reduce these issues. Approach (2) provided only modest benefits compared to those from constituency parsing fusion. The largest LAS increase for (2) is 0.6% for the Stanford Parser, though for Berkeley and Self-trained BLLIP, dependency fusion results in small losses (-0.1% LAS). Two possible reasons are that the dependency baseline is higher than its constituency counterpart and some dependency graphs from the n-best list are duplicates which lowers diversity and may need special handling, but this remains an open question.

While fusion helps on top of a self-trained parser, we also explored whether a fused parser can self-train [68]. To test this, we (1) parsed two million sentences with BLLIP (trained on WSJ), (2) fused those parses, (3) added the fused parses to the gold training set, and (4) retrained the parser on the expanded training. The resulting model did not perform better than a self-trained parsing model that didn't use fusion.

4.4 Conclusions

We presented a simple extension to parse hybridization which adapts model combination techniques to operate over a single parser's n-best list instead of across multiple parsers. By weighting each parse by its probability from the n-best parser, we are able to better capture the confidence at the constituent level. Our best configuration obtains state-of-the-art accuracy on WSJ with an F_1 of 92.6%. This is similar to the accuracy obtained from actual model combination techniques but at a fraction of the computational cost. Additionally, improvements are not limited to a single parser or domain. Fusion improves parser accuracy for six n-best parsers both in-domain and out-of-domain.

Chapter 5

Parsing as Language Modeling

5.1 Introduction

Recent work on deep learning syntactic parsing models has achieved notably good results, e.g., with 92.4 F_1 [31] on Penn Treebank constituency parsing and with 92.8 F_1 [103] . In this theis we borrow from the approaches of both of these works and present a neural-net parse reranker that achieves very good results, 93.8 F_1, with a comparatively simple architecture.

In the remainder of this chapter we outline the major difference between this and previous work — viewing parsing as a language modeling problem. Chapter 5.2 looks more closely at three of the most relevant previous papers. We then describe our exact model (Chapter 5.3), followed by the experimental setup and results (Chapters 5.4 and 5.5).

5.1.1 Language Modeling

Formally, a language model (LM) is a probability distribution over strings of a language:

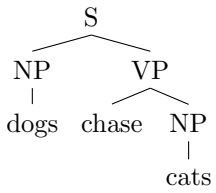
$$\begin{aligned} P(\boldsymbol{x}) &= P(x_1, \cdots, x_n) \\ &= \prod_{t=1}^{n} P(x_t | x_1, \cdots, x_{t-1}), \end{aligned} \tag{5.1}$$

where $\boldsymbol{x}$ is a sentence and t indicates a word position. The efforts in language modeling go into computing $P(x_t|x_1, \cdots, x_{t-1})$, which as described next is useful for parsing as well.

5.1.2 Parsing as Language Modeling

A generative parsing model parses a sentence ($\boldsymbol{x}$) into its phrasal structure ($\boldsymbol{y}$) according to

$$\underset{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})}{\operatorname{argmax}} P(\boldsymbol{x}, \boldsymbol{y}'),$$



(a)

(S (NP dogs)$_{\text{NP}}$ (VP chase (NP cats)$_{\text{NP}}$)$_{\text{VP}}$)$_{\text{S}}$

(b)

Figure 5.1: A tree (a) and its sequential form (b).

where $\mathcal{Y}(\boldsymbol{x})$ lists all possible structures of $\boldsymbol{x}$. If we think of a tree $(\boldsymbol{x}, \boldsymbol{y})$ as a sequence $(\boldsymbol{z})$ [103] as illustrated in Figure 5.1b, we can define a probability distribution over $(\boldsymbol{x}, \boldsymbol{y})$ as follows:

$$\begin{aligned} P(\boldsymbol{x}, \boldsymbol{y}) = P(\boldsymbol{z}) &= P(z_1, \cdots, z_m) \\ &= \prod_{t=1}^{m} P(z_t | z_1, \cdots, z_{t-1}), \end{aligned} \tag{5.2}$$

which is equivalent to Equation (5.1). We have reduced parsing to language modeling and can use language modeling techniques of estimating $P(z_t|z_1, \cdots, z_{t-1})$ for parsing.

5.2 Previous Work

We look here at three neural net (NN) models closest to our research along various dimensions. The first [105] gives the basic language modeling architecture that we have adopted, while the other two [103, 31] are parsing models that have the current best results in NN parsing.

5.2.1 LSTM-LM

The LSTM-LM [105] turns $(x_1, \cdots, x_{t-1})$ into h_t, a hidden state of an LSTM [48, 38, 41], and uses h_t to guess x_t:

$$\begin{aligned} P(x_t | x_1, \cdots, x_{t-1}) &= P(x_t | h_t) \\ &= \text{softmax}(W h_t)[x_t], \end{aligned}$$

where W is a parameter matrix and $[i]$ indexes ith element of a vector. The simplicity of the model makes it easily extendable and scalable, which has inspired a character-based LSTM-LM that works well for many languages [56] and an ensemble of large LSTM-LMs for English with astonishing perplexity of 23.7 [52]. In this thesis, we build a parsing model based on the LSTM-LM [105].

5.2.2 Seq2seq Parser

Researchers [103] observe that a phrasal structure ($\boldsymbol{y}$) can be expressed as a sequence and build a sequence-to-sequence (Seq2Seq) parser, which translates $\boldsymbol{x}$ into $\boldsymbol{y}$ using a conditional probability:

$$\begin{aligned} P(\boldsymbol{y}|\boldsymbol{x}) &= P(y_1, \cdots, y_l|\boldsymbol{x}) \\ &= \prod_{t=1}^{l} P(y_t|\boldsymbol{x}, y_1, \cdots, y_{t-1}), \end{aligned}$$

where the conditioning event $(\boldsymbol{x}, y_1, \cdots, y_{t-1})$ is modeled by an LSTM encoder and an LSTM decoder. The encoder maps $\boldsymbol{x}$ into $\boldsymbol{h}^e$, a set of vectors that represents $\boldsymbol{x}$, and the decoder obtains a summary vector (h'_t) which is concatenation of the decoder's hidden state (h^d_t) and weighted sum of word representations ($\sum_{i=1}^{n} \alpha_i h^e_i$) with an alignment vector ($\boldsymbol{\alpha}$). Finally the decoder predicts y_t given h'_t. Inspired by Seq2Seq parser, our model processes sequential trees.

5.2.3 RNNG

Recurrent Neural Network Grammars (RNNG), a generative parsing model, defines a joint distribution over a tree in terms of actions the model takes to generate the tree [31]:

$$P(\boldsymbol{x}, \boldsymbol{y}) = P(\boldsymbol{a}) = \prod_{t=1}^{m} P(a_t|a_1, \cdots, a_{t-1}), \tag{5.3}$$

where $\boldsymbol{a}$ is a sequence of actions whose output precisely matches the sequence of symbols in $\boldsymbol{z}$, which implies Equation (5.3) is the same as Equation (5.2). RNNG and our model differ in how they compute the conditioning event $(z_1, \cdots, z_{t-1})$: RNNG combines hidden states of three LSTMs that keep track of actions the model has taken, an incomplete tree the model has generated and words the model has generated whereas our model uses one LSTM's hidden state as shown in the next section.

5.3 Model

Our model, the model [105] applied to sequential trees and we call LSTM-LM from now on, is a joint distribution over trees:

$$\begin{aligned} P(\boldsymbol{x}, \boldsymbol{y}) = P(\boldsymbol{z}) &= \prod_{t=1}^{m} P(z_t|z_1, \cdots, z_{t-1}) \\ &= \prod_{t=1}^{m} P(z_t|h_t) \\ &= \prod_{t=1}^{m} \text{softmax}(W h_t)[z_t], \end{aligned}$$

where h_t is a hidden state of an LSTM. Due to lack of an algorithm that searches through an exponentially large phrase-structure space, we use an n-best parser to reduce $\mathcal{Y}(\boldsymbol{x})$ to $\mathcal{Y}'(\boldsymbol{x})$, whose size is polynomial, and use LSTM-LM to find $\boldsymbol{y}$ that satisfies

$$\underset{\boldsymbol{y}' \in \mathcal{Y}'(\boldsymbol{x})}{\operatorname{argmax}} P(\boldsymbol{x}, \boldsymbol{y}'). \tag{5.4}$$

5.3.1 Hyper-parameters

The model has three LSTM layers with 1,500 units and gets trained with truncated backpropagation through time with mini-batch size 20 and step size 50. We initialize starting states with previous mini-batch's last hidden states [99]. The forget gate bias is initialized to be one [53] and the rest of model parameters are sampled from $\mathcal{U}(-0.05, 0.05)$. Dropout is applied to non-recurrent connections [86] and gradients are clipped when their norm is bigger than 20 [82]. The learning rate is $0.25 \cdot 0.85^{\max(\epsilon - 15,\, 0)}$ where ϵ is an epoch number. For simplicity, we use vanilla softmax over an entire vocabulary as opposed to hierarchical softmax [74] or noise contrastive estimation [44].

5.4 Experiments

We describe datasets we use for evaluation, detail training and development processes.[1]

5.4.1 Data

We use the Wall Street Journal (WSJ) of the Penn Treebank [67] for training (2-21), development (24) and testing (23) and millions of auto-parsed "silver" trees [68, 50, 103] for tri-training. To obtain silver trees, we parse the entire section of the New York Times (NYT) of the fifth Gigaword [81] with a product of eight Berkeley parsers [83][2] and ZPar [109] and select 24 million trees on which both parsers agree [64]. We do not resample trees to match the sentence length distribution of the NYT to that of the WSJ [103] because in preliminary experiments Charniak parser [14] performed better when trained on all of 24 million trees than when trained on resampled two million trees.

Given $\boldsymbol{x}$, we produce $\mathcal{Y}'(\boldsymbol{x})$, 50-best trees, with Charniak parser and find $\boldsymbol{y}$ with LSTM-LM as other researchers [31] do with their discriminative and generative models.[3]

5.4.2 Training and Development

Supervision

We unk words that appear fewer than 10 times in the WSJ training (6,922 types) and drop activations with probability 0.7. At the beginning of each epoch, we shuffle the order of trees in the training data. Both perplexity and F_1 of LSTM-LM (G) improve and then plateau (Figure 5.2). Perplexity,

[1] The code and trained models used for experiments are available at github.com/cdg720/emnlp2016.
[2] We use the reimplementation [50].
[3] The discriminative model [31] performs comparably to Charniak (89.8 vs. 89.7).

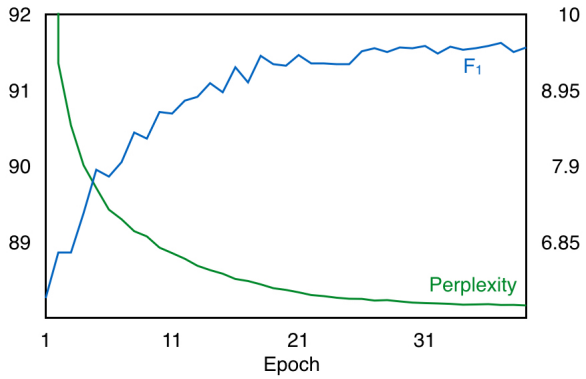


Figure 5.2: Perplexity and F_1 on the development set at each epoch during training.

n	Oracle	Final	Exact
10	94.0	91.2	39.1
50	95.9	91.7	40.0
51^o	100	93.9	49.7
100	96.3	91.7	39.9
500	97.0	91.8	40.0

Table 5.1: The performance of LSTM-LM (G) with varying n-best parses on the dev set. Oracle refers to Charniak parser's oracle F_1. Final and Exact report LSTM-LM (G)'s F_1 and exact match percentage respectively. To simulate an optimal scenario, we include gold trees to 50-best trees and rerank them with LSTM-LM (G) (51^o).

the model's training objective, nicely correlates with F_1, what we care about. Training takes 12 hours (37 epochs) on a Titan X. We also evaluate our model with varying n-best trees including optimal 51-best trees that contain gold trees (51^o). As shown in Table 5.1, the LSTM-LM (G) is robust given sufficiently large n, i.e. 50, but does not exhibit its full capacity because of search errors in Charniak parser. We address this problem in Chapter 5.5.3.

Semi-supervision

We unk words that appear at most once in the training (21,755 types). We drop activations with probability 0.45, smaller than 0.7, thanks to many silver trees, which help regularization. We train LSTM-LM (GS) on the WSJ and a different set of 400,000 NYT trees for each epoch except for the last one during which we use the WSJ only. Training takes 26 epochs and 68 hours on a Titan X. LSTM-LM (GS) achieves 92.5 F_1 on the development.

	Base	Final
Vinyals et al. (2015) [103]	88.3	90.5
Dyer et al. (2016) [31]	89.8	92.4
LSTM-LM (G)	89.7	**92.6**

Table 5.2: F_1 of models trained on WSJ. Base refers to performance of a single base parser and Final that of a final parser.

5.5 Results

5.5.1 Supervision

As shown in Table 5.2, with 92.6 F_1 LSTM-LM (G) outperforms an ensemble of five MTPs [103] and RNNG [31], both of which are trained on the WSJ only.

5.5.2 Semi-supervision

We compare LSTM-LM (GS) to two very strong semi-supervised NN parsers: an ensemble of five MTPs trained on 11 million trees of the high-confidence corpus[4] (HC) [103]; and an ensemble of six one-to-many sequence models trained on the HC and 4.5 millions of English-German translation sentence pairs [65]. We also compare LSTM-LM (GS) to best performing non-NN parsers in the literature. Parsers' parsing performance along with their training data is reported in Table 5.3. LSTM-LM (GS) outperforms all the other parsers with 93.1 F_1.

5.5.3 Improved Semi-supervision

Due to search errors – good trees are missing in 50-best trees – in Charniak (G), our supervised and semi-supervised models do not exhibit their full potentials when Charniak (G) provides $\mathcal{Y}'(\boldsymbol{x})$. To mitigate the search problem, we tri-train Charniak (GS) on all of 24 million NYT trees in addition to the WSJ, to yield $\mathcal{Y}'(\boldsymbol{x})$. As shown in Table 5.3, both LSTM-LM (G) and LSTM-LM (GS) are affected by the quality of $\mathcal{Y}'(\boldsymbol{x})$. A single LSTM-LM (GS) together with Charniak (GS) reaches 93.6 and an ensemble of eight LSTM-LMs (GS) with Charniak (GS) achieves a new state of the art, 93.8 F_1. When trees are converted to Stanford dependencies,[5] UAS and LAS are 95.9% and 94.1%,[6] more than 1% higher than those of the state of the art dependency parser [3]. Why an indirect method (converting trees to dependencies) is more accurate than a direct one (dependency parsing) remains unanswered [59].

[4]The HC consists of 90,000 gold trees, from the WSJ, English Web Treebank and Question Treebank, and 11 million silver trees, whose sentence length distribution matches that of the WSJ, parsed and agreed on by Berkeley parser and ZPar.

[5]Version 3.3.0.

[6]We use the CoNLL evaluator available through the CoNLL website: ilk.uvt.nl/conll/software/eval.pl. Following the convention, we ignore punctuation.

	Base	Oracle	Final	Gold	Silver
Huang et al. (2010) [50]	-	-	92.8	WSJ (40K)	BLLIP (1.8M)
Shindo et al. (2012) [94]	-	-	92.4	WSJ (40K)	-
Choe et al. (2016) [18]	-	-	92.6	WSJ (40K)	NYT (2M)
Vinyals et al. (2015) [103]	-	-	92.8	HC (90K)	HC (11M)
Luong et al. (2016) [65]	-	-	93.0	HC (90K)	HC (11M)
Charniak (G) + LSTM-LM (G)	89.7	96.7	92.6	WSJ (40K)	-
Charniak (G) + LSTM-LM (GS)	89.7	96.7	93.1	WSJ (40K)	NYT (0/10M)
Charniak (GS) + LSTM-LM (G)	91.2	97.1	92.9	WSJ (40K)	NYT (24M/0)
Charniak (GS) + LSTM-LM (GS)	91.2	97.1	93.6	WSJ (40K)	NYT (24M/10M)
Charniak (GS) + E(LSTM-LMs (GS))	91.2	97.1	**93.8**	WSJ (40K)	NYT (24M/11.2M)

Table 5.3: Evaluation of models trained on the WSJ and additional resources. Note that the numbers [103, 65] are not directly comparable as their models are evaluated on OntoNotes-style trees instead of PTB-style trees. E(LSTM-LMs (GS)) is an ensemble of eight LSTM-LMs (GS). X/Y in Silver column indicates the number of silver trees used to train Charniak parser and LSTM-LM. For the ensemble model, we report the maximum number of trees used to train one of LSTM-LMs (GS).

5.6 Conclusion

The generative parsing model we presented in this chapter is very powerful. In fact, we see that a generative parsing model, LSTM-LM, is more effective than discriminative parsing models [31]. We suspect building large models with character embeddings would lead to further improvement as in language modeling [56, 52].

Chapter 6

Left-corner and Right-corner Sequential Parsing

6.1 Introduction

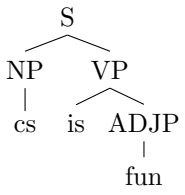
In recent years, many strong greedy constituency parsers have been proposed thanks to the advent of deep learning [103, 19, 24, 31]. We present an extension of the sequence-to-sequence (Seq2Seq) parser [103], which is simple, scalable and fast. When our parser is trained on left-corner (or right-corner) sequential trees, it performs competitively. With an ensemble of semi-supervised rerankers [17], our semi-supervised parsers reach 94.7 F_1.

6.1.1 Seq2Seq Parsing

Sequence-to-sequence models have been applied to machine translation [100, 16, 6]. Vinyals et al. [103] apply a variant of it to syntactic parsing by using word tokens as the input sequence and sequential (linearized) trees as the output (Figure 6.1b). Essentially, the parser learns to "translate" input tokens into an ASCII-based tree format. In this paper, we show that Seq2Seq parsers are more effective when the output sequence consists of left-corner or right-corner sequential trees, described below.

6.1.2 Left (and Right)-corner Parsing

The Penn Treebank (PTB) sequence use to train their Seq2Seq parser [103] is one of many possible representations of trees. PTB sequences are a reasonable default because constituency trees are often annotated in PTB sequence [67]. To develop accurate parsers, we explore two simple alternatives: left-corner and right-corner sequences. At a high level, given a node, a left-corner (LC) transform [87, 2] goes through its left child, its node and the rest of its children from left to right. We can think of



(a)

```
(S (NP cs ) (VP is (ADJP fun ) ) )
```

(b)

```
cs (NP ) (S is (VP fun (ADJP ) ) )
```

(c)

```
fun (ADJP ) (VP is ) (S cs (NP ) )
```

(d)

Figure 6.1: Parse tree (a), its Penn Treebank sequence (b) and corresponding left and right-corner sequences (c, d).

an LC sequence as a rearrangement of a PTB sequence using in-order traversal[1] over a tree. These types of transformations may decrease cognitive load and have been explored from a psycholinguistics perspective [92, 104]. Figure 6.1c illustrates an example LC sequence.

Right-corner (RC) transforms are mirrored versions of LC (see Figure 6.1d for an example).

6.2 Sequential Trees

Because we are representing trees (which are multi-dimensional objects) in one dimension, not all possible sequences are valid trees. For example, any PTB sequence must start with one of 26 open parentheses (e.g., '(NP') and an LC sequence always starts with the first token in the sentence. We claim LC sequences are easier for parsers to learn than PTB ones because LC parsing on average allows fewer symbols than PTB parsing does (Table 6.1). Specifically LC parsing only allows 19.7 symbols on average whereas PTB parsing 27.2.[2] We compute these numbers by following gold sequences in the Wall Street Journal training set and counting numbers of possible symbols along the sequences and averaging them. In other words, the search space for LC parsing is much smaller than that of PTB parsing. Also on average (sub-)trees have shorter spans in LC parsing (14.2) than in PTB parsing (17.9). Our parser is based on LSTMs which can handle some long distance dependencies, but have their limits and has an easier time learning LC sequences than PTB ones.

[1] In parsing, we generally deal with non-binary trees and thus we define the visiting order to be first child, node and remaining children.

[2] 28 symbols in total: 26 start phrase tags, WORD and ')'.

	PTB	LC	RC
Avg. # Valid Symbols	27.2	19.7	19.7
Avg. Distance	17.9	14.2	6.8

Table 6.1: Statistics of sequential trees in WSJ training. The first row shows the average number of valid symbols in three kinds of gold sequential trees. The second one the average distance between pairs of matching parentheses.

6.3 Model

In this section, we detail our model and compare it to the Seq2Seq parser [103]. Capital letters (W_1, W_2) denote matrices, bold lowercase letters $(\boldsymbol{f}, \boldsymbol{b})$ vectors and lowercase letters (t, s) numbers.

6.3.1 Encoder

We replace the LSTM [48, 38] encoder of the Seq2Seq parser with a bidirectional LSTM (BLSTM) [42] as BLSTMs have been proven to be very effective for modeling words and their contexts for parsing [57, 24, 29]. Given a sequence of word embeddings $(\boldsymbol{e}_1, \cdots, \boldsymbol{e}_n)$, the BLSTM provides $(\boldsymbol{f}_1, \cdots, \boldsymbol{f}_n)$ and $(\boldsymbol{b}_1, \cdots, \boldsymbol{b}_n)$ where $\boldsymbol{f}_i$ and $\boldsymbol{b}_i$ are hidden states at position i from forward and backward LSTMs, respectively. As word representations, the decoder (described below) uses the concatenation of these, $\boldsymbol{v}_i = \boldsymbol{f}_i \oplus \boldsymbol{b}_i$, which represents word at position i and its surrounding words. The encoder has an auxiliary objective over word representations to predict part-of-speech (POS) tags so that the encoder includes tagging information inside these word representations, which help parsing a bit. The objective is defined as

$$P(t_i|\boldsymbol{v}_i) = \text{softmax}(W_1 \cdot \boldsymbol{v}_i + \boldsymbol{b}_1)[\text{Idx}(t_i)], \tag{6.1}$$

where t_i is a gold tag for word at position i and Idx is a function that returns the index of t_i.

6.3.2 Decoder

As in shift-reduce parsing [78], we maintain a buffer of word representations for the input sentence, $(\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n)$, from left to right and pop one word from the buffer whenever the decoder outputs a word. A key difference from shift-reducing parsing is that instead of representing parsing state (e.g., previous symbols taken) with a discrete stack, we use an LSTM. At step j, the decoder reads in an embedding of its previous output, $\boldsymbol{p}_{j-1}$, and the first word representation in the buffer, $\boldsymbol{v}_{x_j}$, where x_j is the position of the first word in the buffer at step j. The decoder pushes $\boldsymbol{p}_{j-1} \oplus \boldsymbol{v}_{x_j}$, concatenation of the two, through its LSTM function and uses its hidden state, $\boldsymbol{h}_j$, to predict a symbol s_j:[3]

$$P(s_j|\boldsymbol{h}_j) = \text{softmax}(W_2 \cdot \boldsymbol{h}_j + \boldsymbol{b}_2)[\text{Idx}(s_j)], \tag{6.2}$$

[3]In pilot experiments, we found that processing the previous output and first word together worked better than processing the output only and later passing the hidden state along with word representation to softmax layers.

where s_j is a gold symbol at position j. During training, we optimize the sum of Equations (6.1) and (6.2), but during inference, we only use Equation (6.2). The decoder predicts one of 26 phrase tags, ')', WORD, END and PAD.[4] When the decoder predicts WORD, it outputs the first word in the buffer rather than the WORD symbol.

Unlike [103], we don't use attention over word representations because in parsing we know what word to process next. Compared to theirs, ours can be thought of as a Seq2Seq with explicit attention.

6.3.3 Network Architecture

We use 300 dimensions for the BLSTM encoder, 900 for the LSTM decoder. For both encoder and decoder, we stack two LSTMs and have skip connections from input layers to second LSTM layers [45].[5] We apply dropout [97] to embeddings and vertical LSTM connections [86].

6.4 Experiments

We use the standard split of Wall Street Journal (WSJ) in the Penn Treebank for training (Sections 2–21), validating (22) and testing (23) our parsers. We replace singleton tokens in the training with UNK and remove spurious unary chains, e.g., '(NP (NP a dog))' becomes '(NP a dog)', during training but keep them during evaluation for comparison. For semi-supervised experiments [68], we train the parsers on auto-parsed New York Times (NYT) trees [17] in addition to WSJ. Models and code are available at `github.com/cdg720/emnlp2017`.

6.4.1 Supervised Training

We train models for 50 epochs and save parameters that perform best on Section 22. During training, models attempt to maximize the probability of symbols in gold sequential trees. During inference, they greedily choose the most likely valid symbols. Training takes 5.3 hours on an Nvidia 1080 GPU and parsing Section 22 takes a little less than 13 seconds (130 sentences per second). We report the hyperparameters in the Chapter 6.4.4.

The performance of our models with different sequential trees is reported in Table 6.2. Both LC and RC perform substantially better than PTB representations. During decoding, LC (WSJ) and RC (WSJ) can get confused and produce infinite chains of unaries. In these cases, we stop decoding and output trees with single S nodes. Sampling a few trees and selecting one for a failed sentence easily solves the problem. Semi-supervised training also improves the model, solving this problem as described below.

[4] END denotes the end of parsing and PAD gets added to short sentences for batching.

[5] For example, the input to the second layer's forward LSTM at position i is $\boldsymbol{e}_i + \boldsymbol{f}_i^1$ where $\boldsymbol{f}_i^1$ is the hidden state of the first layer's forward LSTM at position i.

Form	F_1 (PF)
LC (WSJ)	91.5 (1)
RC (WSJ)	**91.6** (2)
PTB (WSJ)	91.0 (0)
LC (semi)	**93.3** (0)
RC (semi)	93.2 (0)

Table 6.2: Performance and number of failed parses (PF) on Section 22.

α	Oracle	# Unique	Reranked
0.5	**99.1**	**321.1**	94.6
0.7	99.0	98.0	**94.8**
0.9	98.6	43.0	**94.8**
1.1	98.3	24.7	94.7

Table 6.3: Oracle and reranked performance of LC (semi) and CC (semi), varying α on WSJ Section 22. 1000 trees are sampled per sentence.

6.4.2 Semi-supervised Training

One advantage of sequential models is that they are very scalable and we train our models on millions of NYT trees. Training data consists of WSJ training and 1.2 million auto-parsed NYT trees (resampled every epoch). As in supervised training, we train the models for 50 epochs and choose best settings on the validation set. The semi-supervised training takes about 83.3 hours on a single GPU. See results in Table 6.2.

6.4.3 Sampling and Reranking

One caveat of greedy parsing is the model makes a series of local decisions, which obviously aren't globally optimal. To overcome this, as in [31] we sample N trees with our parser for each sentence. For each tree, we sample a symbol at a time from a renormalized distribution with exponentiation α. We feed the samples through the CC reranker[6] [17] and evaluate the reranked trees. The results of varying α and N are reported in Tables 6.3 and 6.4. First note that the oracle scores are remarkable and there is a much room for developing better rerankers. α seems to matter a bit but N doesn't (at least once sufficiently large). For supervised evaluation, we sample 100 trees for each sentence to compare our model to reranking parsers of [31] and [62]. For semi-supervised evaluation, we sample 200 trees.

6.4.4 Hyperparameters

For supervised training, we run three grid searches on the validation set for LC, RC and PTB and select the best setting for each. We initialize all parameters with uniform distribution between -0.03 and 0.03 and use stochastic gradient descent with momentum (0.99) and initial learning rate (0.01) and batch size 50. We decay the learning rate with the following: $0.01 \cdot \lambda^{\max(\epsilon-20,0)}$ where λ is a

[6]Models from `github.com/cdg720/emnlp2016`.

N	Oracle	# Unique	Reranked
100	98.2	18.9	**94.8**
200	98.5	30.7	**94.8**
500	98.8	59.1	**94.8**
1000	**99.0**	**98.0**	**94.8**

Table 6.4: Oracle and reranked performance of LC (semi) and CC (semi), varying the number of samples on WSJ Section 22. $\alpha = 0.7$ is used.

Parameter	LC	RC	PTB
dropout	0.55	0.55	0.5
clipping	5	10	5
decaying (λ)	0.85	0.85	0.8
exponentiation (α)	0.75	0.85	-

Table 6.5: Hyperparameters

decaying parameter and ϵ is an epoch number. The rest of the parameters are reported in Table 6.5.

The grid search is harder for semi-supervised training because it takes longer to finish training and we use the same setting for training both LC and RC: dropout (0.4), clipping (5), decaying (0.8) and batch size 100 for NYT trees. For sampling, we use $\alpha = 0.7$ for LC and $\alpha = 0.65$ for RC.

6.4.5 Supervised Evaluation

We first evaluate our parsers and compare them to strong neural greedy parsers and then compare our parsers with CC (WSJ) to other reranking parsers on Section 23. As shown in Table 6.6, the reduction in search space seems helpful for parsing but the reduction in the average distance between parentheses doesn't matter as much. We leave exploration in this area to future work. Our greedy parsers (LC and RC) are comparable to other greedy parsers [19, 24, 31]. Together with CC (WSJ), LC and RC are comparable to [31] and a little worse than [62].

6.4.6 Semi-supervised Evaluation

LC (semi) and RC (semi) perform comparably to state-of-the-art semi-supervised parsers. With CC (semi), LC (semi) and RC (semi) exhibit strong performance: 94.2 and 94.3. We can combine the samples of LC (semi) and RC (semi) and rerank them with a reranker (94.5) and five rerankers (94.7). Converting to dependencies[7], we get 94.6 LAS and 96.4 UAS, which is within the range of human interannotator agreement.[8] The system of two parsers and five rerankers achieves 88.9 on BNC [35], 90.6 on Brown [36], 80.8 on GENIA [55], 81.7 on Switchboard [40] and 93.1 on QuestionBank [54]. Out-of-domain parsing remains a hard problem, but these numbers represent significant progress.

[7]Stanford Dependences, version 3.3.0

[8]`research.googleblog.com/2016/05/announcing-syntaxnet-worlds-most.html`

Model	F_1 (PF)
Coavoux and Crabbé (2016) [19]	88.6
Vinyals et al. (2015) [103]	90.5
Cross and Huang (2016) [24]	91.3
Dyer et al. (2016) [31]	91.2
LC (WSJ)	**91.4** (1)
RC (WSJ)	91.2 (2)
PTB (WSJ)	91.0 (0)
Choe and Charniak (2016) [17]	92.6
Dyer et al. (2016) [31]	93.3
Kuncoro et al. (2017) [62]	**93.6**
LC (WSJ) + CC (WSJ)	93.2
RC (WSJ) + CC (WSJ)	93.1

Table 6.6: Comparison of supervised greedy parsers (top) and reranking parsers (bottom) on WSJ Section 23. PF indicates parse failures.

Model	F_1
Vinyals et al. (2015) [103]	92.8
Luong et al. (2016) [65]	**93.0**
LC (semi)	**93.0**
RC (semi)	92.7
Choe and Charniak (2016) [17]	93.8
LC (semi) + CC (semi)	94.2
RC (semi) + CC (semi)	94.3
L/RC (semi) + CC (semi)	94.5
L/RC (semi) + 5 CCs (semi)	**94.7**

Table 6.7: Comparison of semi-supervised parsers (top) and reranking parsers (bottom) on WSJ Section 23.

6.5 Conclusion

We have shown simple sequential models are effective for constituency parsing. Using left-corner transformations dramatically reduces the search space allowing the parser to parse accurately in linear-time without pre-training, an external POS tagger or hand-engineered features. Using semi-supervised training and reranking, our accuracies on six domains are state-of-the-art.

Chapter 7

Conclusion

In this thesis, we have shown that simple sequential models developed for language modeling and machine translation can be effectively used for parsing. Specifically, on a domain of new articles, our system of reranking parsers achieves a human-level accuracy, which has been the goal of parsing research for the past few decades. Given a sentence, our parser can generate a list of trees, one of which is identical or very close to its gold tree (Chapter 6) and our reranker can reliably find a high-quality tree in the list (Chapter 5). Our system has harder time parsing out-of-domain sentences but shows strong results on these domains as well even though it is not trained on them. Given enough gold parse trees in these domains, we suspect our system would perform very well on these domains as well.

For domains (or languages) with small training data, the method of generating parse trees using our method of parsing paraphrases jointly can be used in addition to the standard parse generation methods to gather a large number of high-quality parse trees (Chapter 3). The fusion method described in Chapter 4 can be effectively applied to these low-quality parse trees to improve results. Given the rapid progress the parsing community has made in the field, we believe it will see general human-level parsers in a near future.

Appendix A

Appendix

We list a few examples of sentences that our reranking parsing model fails to parse correctly. Top ones are gold and bottom ones are from our model.

(S (PP On (NP Friday)) , (NP mortgage issues) (VP gained (ADVP (ADVP as much) (PP as (NP (QP 1 5/32))))) .)
(S (PP On (NP Friday)) , (NP mortgage issues) (VP gained (NP (QP as much as 1 5/32))) .)
(S (NP The bill) (VP intends (S (VP to (VP restrict (NP the RTC) (PP to (NP Treasury borrowings only))))) , (SBAR unless (S (NP the agency) (VP receives (NP specific congressional authorization))))) .)
(S (NP The bill) (VP intends (S (VP to (VP restrict (NP the RTC) (PP to (NP Treasury borrowings)) (ADVP only) , (SBAR unless (S (NP the agency) (VP receives (NP specific congressional authorization)))))))) .)
(FRAG (ADVP Now) , (PP as (PP for (NP tomorrow))) , (INTJ hell) , (SBARQ (WHNP who) (SQ (VP knows))) ?)
(FRAG (ADVP Now) , (PP as (PP for (NP tomorrow))) , (NP (NP hell) , (SBAR (WHNP who) (S (VP knows)))) ?)
(S (NP The Columbia economists) (ADVP also) (VP have (VP reconstructed (SBAR (WHADVP how) (S (NP the long leading index) (VP would (VP have (VP behaved , (SBAR (SINV had (NP it) (VP existed))) (PRN , (PP in (NP 1929)) ,) (PP before (NP (NP the stock market crash) (PP in (NP October)) (SBAR (WHNP that) (S (VP ushered (PRT in) (NP the Great Depression)))))))))))) .)
(S (NP The Columbia economists) (ADVP also) (VP have (VP reconstructed (SBAR (WHADVP how) (S (NP the long leading index) (VP would (VP have (VP behaved))))) , (SBAR (SINV had (NP it) (VP existed , (PP in (NP 1929)) , (PP before (NP (NP the stock market crash) (PP in (NP October)) (SBAR (WHNP that) (S (VP ushered (PP in (NP the Great Depression)))))))))))) .)
(S (NP Bangkok , Manila , Seoul , Taipei and Jakarta) (VP escaped (PP with (NP (ADJP slightly smaller) losses))) .)
(S (NP (NP Bangkok) , (NP Manila) , (NP Seoul) , (NP Taipei) and (NP Jakarta)) (VP escaped (PP with (NP (ADJP slightly smaller) losses))) .)

(S (PP At (NP Shearson Lehman)) , (NP executives) (VP (VP created (NP potential new commercials) (UCP (NP Friday night) and (PP throughout (NP the weekend)))) , (VP (ADVP then) had (S (VP to (VP regroup (NP yesterday afternoon)))))) .)
(S (PP At (NP Shearson Lehman)) , (NP executives) (VP (VP created (NP potential new commercials) (UCP (NP Friday night) and (PP throughout (NP the weekend)))) , then (VP had (S (VP to (VP regroup))) (NP yesterday afternoon))) .)
(S (NP Gold) (VP (ADVP also) rose) .)
(S (NP Gold) (ADVP also) (VP rose) .)
(S (NP It) (VP (VP (ADVP once) pushed (PP for (NP (NP a national housing production goal) (VP set (PP by (NP the federal government)))))) and (VP has (VP (ADVP regularly) advanced (NP anti-recession housing measures)))) .)
(S (NP It) (ADVP once) (VP (VP pushed (PP for (NP (NP a national housing production goal) (VP set (PP by (NP the federal government)))))) and (VP has (ADVP regularly) (VP advanced (NP anti-recession housing measures)))) .)
(S (NP (NP Net) (PP for (NP the third quarter)) , (VP restated) ,) (VP is (NP (NP (QP $ 1.6 million)) , or (NP (NP 10 cents) (NP a share)))) .)
(S (NP (NP Net) (PP for (NP (NP the third quarter) , (ADJP restated) ,))) (VP is (NP (NP (QP $ 1.6 million)) , or (NP (NP 10 cents) (NP a share)))) .)
(S (NP (NP Many U.S. trading operations) , (VP wanting (S (VP to (VP keep (NP a watchful eye) (PP on (NP (NP Japanese trading) (PP as (NP (NP an indication) (PP of (SBAR (WHADVP where) (S (NP U.S. trading) (VP would (VP begin)))))))))))) ,) (VP were (VP (ADVP fully) staffed (PP during (NP the Tokyo trading session)))) .)
(S (NP Many U.S. trading operations) , (S (VP wanting (S (VP to (VP keep (NP (NP a watchful eye) (PP on (NP Japanese trading))) (PP as (NP (NP an indication) (PP of (SBAR (WHADVP where) (S (NP U.S. trading) (VP would (VP begin)))))))))))) , (VP were (ADJP fully staffed) (PP during (NP the Tokyo trading session))) .)
(S (S (NP (NP Security Pacific 's) earnings growth) (VP slowed (PP in (NP the third quarter)))) , but (S (NP the Los Angeles bank holding company) (VP was (ADVP still) (ADJP able (S (VP to (VP post (NP (NP a (ADJP 10 %) increase) (PP in (NP net income))) (PP because of (NP (NP robust growth) (PP in (NP residential real-estate and consumer loans)))))))))) .)
(S (S (NP (NP Security Pacific 's) earnings growth) (VP slowed (PP in (NP the third quarter)))) , but (S (NP the Los Angeles bank holding company) (VP was (ADVP still) (ADJP able (S (VP to (VP post (NP (NP a (ADJP 10 %) increase) (PP in (NP net income)))))) (PP because of (NP (NP robust growth) (PP in (NP residential real-estate and consumer loans)))))) .)
(S (NP (NP The latest reading) (PP of (NP 223.0))) (VP was (ADVP up (PP (PP from (NP 222.3) (PP in (NP July))) and (PP (NP 215.3) (ADVP (ADVP as recently) (PP as (NP March))))))) .)
(S (NP (NP The latest reading) (PP of (NP 223.0))) (VP was (ADVP up (PP from (NP (NP (NP 222.3) (PP in (NP July))) and (NP (NP 215.3) (ADVP (ADVP as recently) (PP as (NP March)))))))) .)
(S (NP (NP Such loans) (PP to (NP Argentina))) (VP (ADVP also) remain (ADJP classified (PP as (ADJP non-accruing))) , (S (VP costing (NP the bank) (NP (NP (QP $ 10 million)) (PP of (NP interest income))) (PP in (NP the third period))))) .)
(S (NP (NP Such loans) (PP to (NP Argentina))) (ADVP also) (VP remain (VP classified (PP as (ADJP non-accruing)) , (S (VP costing (NP the bank) (NP (NP (QP $ 10 million)) (PP of (NP interest income))) (PP in (NP the third period)))))) .)

(NP (NP (ADJP Negotiable , bank-backed) business credit instruments) (VP (ADVP typically) financing (NP an import order)) .)
(NP (NP Negotiable , bank-backed business credit instruments) (VP (ADVP typically) financing (NP an import order)) .)
(S (NP Such features) (VP have (VP been (ADJP especially attractive (PP to (NP (NP (NP professional photographers) and (NP marketing executives)) , (SBAR (WHNP who) (S (VP have (VP been (VP (ADVP steadily) increasing (NP (NP their use) (PP of (NP black and white)) (PP in (NP advertising))))))))))))) .)
(S (NP Such features) (VP have (VP been (ADJP especially attractive (PP to (NP (NP (NP professional photographers) and (NP marketing executives)) , (SBAR (WHNP who) (S (VP have (VP been (VP (ADVP steadily) increasing (NP (NP their use) (PP of (ADJP black and white))) (PP in (NP advertising))))))))))))) .)
(S (NP The Washington Post) (VP reported (SBAR that (S (NP unidentified senior administration officials) (VP (VP were (VP frustrated (PP with (NP (NP (NP Webster 's) low-profile activities) (PP during (NP the insurrection)))))) and (VP wanted (S (NP him) (VP replaced))))))) .)
(S (NP The Washington Post) (VP reported (SBAR that (S (NP unidentified senior administration officials) (VP (VP were (VP frustrated (PP with (NP (NP Webster 's) low-profile activities)) (PP during (NP the insurrection)))) and (VP wanted (S (NP him) (VP replaced))))))) .)
(S (NP Hertz Equipment) (VP is (NP (NP a major supplier) (PP of (NP rental equipment))) (PP in (NP (NP the U.S.) , (NP France) , (NP Spain) and (NP the U.K)))) .)
(S (NP Hertz Equipment) (VP is (NP (NP a major supplier) (PP of (NP rental equipment)) (PP in (NP (NP the U.S.) , (NP France) , (NP Spain) and (NP the U.K))))) .)
(SINV (VP Contributing (PP to (NP the selling pressure))) (VP were) (NP (NP dispatches) (PP by (NP several investment firms)) (VP advising (NP clients) (S (VP to (VP (VP boost (NP their stock holdings)) and (VP reduce (NP (NP the size) (PP of (NP their cash or bond portfolios))))))))) .)
(SINV (VP Contributing (PP to (NP the selling pressure))) (VP were) (NP (NP dispatches) (PP by (NP (NP several investment firms) (VP advising (NP clients) (S (VP to (VP (VP boost (NP their stock holdings)) and (VP reduce (NP (NP the size) (PP of (NP their cash or bond portfolios))))))))))) .)
(S (SBAR As (S (NP the London market) (VP rallied))) , (NP some) (VP wondered (SBAR whether (S (NP (NP the weekend) (PP of (NP worrying and jitters))) (VP had (VP been (PP worth (NP it))))))) .)
(S (SBAR As (S (NP the London market) (VP rallied))) , (NP some) (VP wondered (SBAR whether (S (NP (NP the weekend) (PP of (NP worrying and jitters))) (VP had (VP been (ADJP worth (NP it))))))) .)
(S (NP (NP Export sales) (PP for (NP (NP leisure items) (ADVP alone)))) , (PP for (NP instance)) , (VP totaled (NP (QP 184.74 billion) yen) (PP in (NP the 12 months)) , (ADVP up (PP from (NP (QP 106.06 billion)) (PP in (NP the (ADJP previous fiscal) year))))) .)
(S (NP (NP Export sales) (PP for (NP (NP leisure items) (ADVP alone)))) , (PP for (NP instance)) , (VP totaled (NP (QP 184.74 billion) yen) (PP in (NP the 12 months)) , (ADVP up (PP from (NP 106.06 billion) (PP in (NP the previous fiscal year))))) .)
(S (NP Revenue) (VP was (NP (NP (QP $ 444.9 million)) , (PP including (NP net interest)) ,) (ADVP down slightly (PP from (NP (QP $ 450.7 million))))) .)
(S (NP Revenue) (VP was (NP (NP (QP $ 444.9 million)) , (PP including (NP net interest))) , (ADVP down (ADVP slightly) (PP from (NP (QP $ 450.7 million))))) .)

Bibliography

[1] Anne Abeillé, Lionel Clément, and Franc(c)ois Toussenel. Building a treebank for french. *Treebanks*, 2003.

[2] Alfred V. Aho and Jeffrey D. Ullman. *The theory of parsing, translation, and compiling.* Prentice-Hall, Inc., 1972.

[3] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016.

[4] Sam Anzaroot, Alexandre Passos, David Belanger, and Andrew McCallum. Learning soft linear constraints with application to citation field extraction. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 2014.

[5] Michiel Bacchiani, Michael Riley, Brian Roark, and Richard Sproat. Map adaptation of stochastic grammars. *Computer speech & language*, 2006.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[7] Mohit Bansal and Dan Klein. Web-scale features for full-scale parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, 2011.

[8] Jonathan Berant and Percy Liang. Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 2014.

[9] Rahul Bhagat and Eduard Hovy. What is a paraphrase? *Computational Linguistics*, 2013.

[10] Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. The tiger treebank. In *Proceedings of the workshop on treebanks and linguistic theories*, 2002.

[11] David Burkett, John Blitzer, and Dan Klein. Joint parsing and alignment with weakly synchronized grammars. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2010.

[12] Steven Burrows, Martin Potthast, and Benno Stein. Paraphrase acquisition via crowdsourcing and machine learning. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2013.

[13] Chris Callison-Burch, Philipp Koehn, and Miles Osborne. Improved statistical machine translation using paraphrases. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, 2006.

[14] Eugene Charniak. A maximum-entropy-inspired parser. In *1st Meeting of the North American Chapter of the Association for Computational Linguistics*, 2000.

[15] Eugene Charniak and Mark Johnson. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, 2005.

[16] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, 2014.

[17] Do Kook Choe and Eugene Charniak. Parsing as language modeling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016.

[18] Do Kook Choe, David McClosky, and Eugene Charniak. Syntactic parse fusion. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015.

[19] Maximin Coavoux and Benoit Crabbé. Neural greedy constituent parsing with dynamic oracles. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016.

[20] Trevor Cohn and Mirella Lapata. An abstractive approach to sentence compression. *ACM Transactions on Intelligent Systems and Technology*, 2013.

[21] Michael Collins. Discriminative reranking for natural language parsing. In *Proceedings of the Seventeenth International Conference on Machine Learning*, 2000.

[22] Michael Collins and Terry Koo. Discriminative reranking for natural language parsing. *Computational Linguistics*, 2005.

[23] Gregory Coppola and Mark Steedman. The effect of higher-order dependency features in discriminative phrase-structure parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, 2013.

[24] James Cross and Liang Huang. Span-based constituency parsing with a structure-label system and provably optimal dynamic oracles. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016.

[25] Marie-Catherine De Marneffe, Bill MacCartney, Christopher D. Manning, et al. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, 2006.

[26] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, 2009.

[27] Michael Denkowski and Alon Lavie. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, 2014.

[28] William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proc. of IWP*, 2005.

[29] Timothy Dozat and Christopher D Manning. Deep biaffine attention for neural dependency parsing. *arXiv preprint arXiv:1611.01734*, 2016.

[30] Pablo Ariel Duboue and Jennifer Chu-Carroll. Answering the question you wish they had asked: The impact of paraphrasing for question answering. In *Proceedings of the Human Language Technology Conference of the NAACL*, 2006.

[31] Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A Smith. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016.

[32] Jason M. Eisner. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th conference on Computational linguistics*, 1996.

[33] Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. Paraphrase-driven learning for open question answering. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, 2013.

[34] Victoria Fossum and Kevin Knight. Combining constituent parsers. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, 2009.

[35] Jennifer Foster and Josef van Genabith. Parser evaluation and the BNC: Evaluating 4 constituency parsers with 3 metrics. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, 2008.

[36] Winthrop Nelson Francis and Henry Kučera. *Manual of information to accompany a standard corpus of present-day edited American English, for use with digital computers.* Brown University, Department of Linguistics, 1989.

[37] Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. PPDB: The paraphrase database. In *Proceedings of NAACL-HLT*, 2013.

[38] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *The Journal of Machine Learning Research*, 2003.

[39] Daniel Gildea. Corpus variation and parser performance. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, 2001.

[40] John J. Godfrey, Edward C. Holliman, and Jane McDaniel. Switchboard: Telephone speech corpus for research and development. In *Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on*, 1992.

[41] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

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

[43] Spence Green and Christopher D Manning. Better arabic parsing: Baselines, evaluations, and analysis. In *Proceedings of the 23rd International Conference on Computational Linguistics*, 2010.

[44] Michael U. Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *The Journal of Machine Learning Research*, 2012.

[45] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[46] Michael Heilman and Noah A Smith. Tree edit models for recognizing textual entailments, paraphrases, and answers to questions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2010.

[47] John C. Henderson and Eric Brill. Exploiting diversity in natural language processing: Combining parsers. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 1999.

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

[49] Liang Huang. Forest reranking: Discriminative parsing with non-local features. In *Proceedings of ACL-08: HLT*, 2008.

[50] Zhongqiang Huang, Mary Harper, and Slav Petrov. Self-training with products of latent variable grammars. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, 2010.

[51] Mark Johnson and Ahmet Engin Ural. Reranking the berkeley and brown parsers. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2010.

[52] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

[53] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

[54] John Judge, Aoife Cahill, and Josef van Genabith. Questionbank: Creating a corpus of parse-annotated questions. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, 2006.

[55] J-D Kim, Tomoko Ohta, Yuka Tateisi, and Jun'ichi Tsujii. Genia corpus—a semantically annotated corpus for bio-textmining. *Bioinformatics*, 2003.

[56] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. Character-aware neural language models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[57] Eliyahu Kiperwasser and Yoav Goldberg. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 2016.

[58] Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, 2003.

[59] Lingpeng Kong and Noah A Smith. An empirical comparison of parsing methods for stanford dependencies. *arXiv preprint arXiv:1404.4314*, 2014.

[60] Terry Koo, Xavier Carreras, and Michael Collins. Simple semi-supervised dependency parsing. In *Proceedings of ACL: HLT*, 2008.

[61] Terry Koo and Michael Collins. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, 2010.

[62] Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A Smith. What do recurrent neural network grammars learn about syntax? In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, 2017.

[63] Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. Low-rank tensors for scoring dependency structures. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 2014.

[64] Zhenghua Li, Min Zhang, and Wenliang Chen. Ambiguity-aware ensemble training for semi-supervised dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 2014.

[65] Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. Multi-task sequence to sequence learning. *International Conference on Learning Representations*, 2016.

[66] Mohamed Maamouri, Ann Bies, Tim Buckwalter, and Wigdan Mekki. The penn arabic treebank: Building a large-scale annotated arabic corpus. In *NEMLAR conference on Arabic language resources and tools*, 2004.

[67] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 1993.

[68] David McClosky, Eugene Charniak, and Mark Johnson. Effective self-training for parsing. In *Proceedings of the Human Language Technology Conference of the NAACL*, 2006.

[69] David McClosky, Eugene Charniak, and Mark Johnson. Reranking and self-training for parser adaptation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, 2006.

[70] David McClosky, Eugene Charniak, and Mark Johnson. Automatic domain adaptation for parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2010.

[71] David McClosky, Sebastian Riedel, Mihai Surdeanu, Andrew McCallum, and Christopher D. Manning. Combining joint models for biomedical event extraction. *BMC Bioinformatics*, 2012.

[72] Ryan McDonald and Joakim Nivre. Analyzing and integrating dependency parsers. *Computational Linguistics*, 2011.

[73] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, 2005.

[74] Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, 2005.

[75] Preslav Nakov and Marti Hearst. Using the web as an implicit training set: Application to structural ambiguity resolution. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, 2005.

[76] Shashi Narayan and Shay B. Cohen. Diversity in spectral learning for natural language parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015.

[77] Matteo Negri, Yashar Mehdad, Alessandro Marchetti, Danilo Giampiccolo, and Luisa Bentivogli. Chinese whispers: Cooperative paraphrase acquisition. In *LREC*, 2012.

[78] Joakim Nivre. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 2008.

[79] Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. MaltParser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 2007.

[80] Scott Nowson and Robert Dale. Charting democracy across parsers. In *Proceedings of the Australasian Language Technology Workshop*, 2007.

[81] Robert Parker, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. English gigaword fifth edition. *Linguistic Data Consortium, LDC2011T07*, 2011.

[82] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

[83] Slav Petrov. Products of random latent variable grammars. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2010.

[84] Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, 2006.

[85] Slav Petrov, Pi-Chuan Chang, Michael Ringgaard, and Hiyan Alshawi. Uptraining for accurate deterministic question parsing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, 2010.

[86] Vu Pham, Théodore Bluche, Christopher Kermorvant, and Jérôme Louradour. Dropout improves recurrent neural networks for handwriting recognition. In *2014 14th International Conference on Frontiers in Handwriting Recognition*, 2014.

[87] Stanley J. Rosenkrantz and Philip M. Lewis II. Deterministic left corner parser. In *IEEE Conference Record of the 11th Annual Symposium on Switching and Automata*, 1970.

[88] Alexander M. Rush and Michael Collins. A tutorial on dual decomposition and Lagrangian relaxation for inference in natural language processing. *Journal of Artificial Intelligence Research*, 2012.

[89] Alexander M. Rush, Roi Reichart, Michael Collins, and Amir Globerson. Improved parsing and POS tagging using inter-sentence consistency constraints. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2012.

[90] Alexander M. Rush, David Sontag, Michael Collins, and Tommi Jaakkola. On dual decomposition and linear programming relaxations for natural language processing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, 2010.

[91] Kenji Sagae and Alon Lavie. Parser combination by reparsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, 2006.

[92] William Schuler. Positive results for parsing with a bounded stack using a model-based right-corner transform. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2009.

[93] Djamé Seddah, Sandra Kübler, and Reut Tsarfaty. Introducing the spmrl 2014 shared task on parsing morphologically-rich languages. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, 2014.

[94] Hiroyuki Shindo, Yusuke Miyao, Akinori Fujino, and Masaaki Nagata. Bayesian symbol-refined tree substitution grammars for syntactic parsing. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, 2012.

[95] Wojciech Skut, Brigitte Krenn, Thorsten Brants, and Hans Uszkoreit. An annotation scheme for free word order languages. In *Proceedings of the fifth conference on Applied natural language processing*, 1997.

[96] Richard Socher, John Bauer, Christopher D. Manning, and Ng Andrew Y. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, 2013.

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

[98] Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. The CoNLL-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*, 2008.

[99] Ilya Sutskever. *Training recurrent neural networks*. PhD thesis, University of Toronto, 2013.

[100] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27*, 2014.

[101] Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, 2003.

[102] Martin Tschirsich and Gerold Hintz. Leveraging crowdsourcing for paraphrase recognition. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, 2013.

[103] Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. Grammar as a foreign language. In *Advances in Neural Information Processing Systems 28*, 2015.

[104] Stephen Wu, Asaf Bachrach, Carlos Cardenas, and William Schuler. Complexity metrics in an incremental right-corner parser. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, 2010.

[105] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[106] Daniel Zeman and Zdeněk Žabokrtskỳ. Improving parsing accuracy by combining diverse dependency parsers. In *Proceedings of the Ninth International Workshop on Parsing Technology*, 2005.

[107] Hao Zhang and Ryan McDonald. Enforcing structural diversity in cube-pruned dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 2014.

[108] Hui Zhang, Min Zhang, Chew Lim Tan, and Haizhou Li. K-best combination of syntactic parsers. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, 2009.

[109] Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. Fast and accurate shift-reduce constituent parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, 2013.