Ruprecht-Karls-Universität Heidelberg
Neuphilologische Fakultät

Neural Techniques for German Dependency Parsing

Bich-Ngoc Do

Institut für Computerlinguistik

This dissertation is submitted for the degree of
DOCTOR OF PHILOSOPHY

11th November 2023

Title:	Neural Techniques for German Dependency Parsing
Author:	Bich-Ngoc Do
Faculty/Institute:	Institut für Computerlinguistik Neuphilologische Fakultät
Type:	Dissertation
Supervisor:	Dr. Ines Rehbein Data and Web Science Group Universität Mannheim
Second supervisor:	Prof. Dr. Anette Frank Institut für Computerlinguistik Universität Heidelberg
Date of Submission:	19th July 2021
Date of Defense:	24th February 2022

To my family

Abstract

Syntactic parsing is the task of analyzing the structure of a sentence based on some predefined formal assumption. It is a key component in many natural language processing (NLP) pipelines and is of great benefit for natural language understanding (NLU) tasks such as information retrieval or sentiment analysis. Despite achieving very high results with neural network techniques, most syntactic parsing research pays attention to only a few prominent languages (such as English or Chinese) or language-agnostic settings. Thus, we still lack studies that focus on just one language and design specific parsing strategies for that language with regards to its linguistic properties.

In this thesis, we take German as the language of interest and develop more accurate methods for German dependency parsing by combining state-of-the-art neural network methods with techniques that address the specific challenges posed by the language-specific properties of German. Compared to English, German has richer morphology, semi-free word order, and case syncretism. It is the combination of those characteristics that makes parsing German an interesting and challenging task.

Because syntactic parsing is a task that requires many levels of language understanding, we propose to study and improve the knowledge of parsing models at each level in order to improve syntactic parsing for German. These levels are: (sub)word level, syntactic level, semantic level, and sentence level.

At the (sub)word level, we look into a surge in out-of-vocabulary words in German data caused by *compounding*. We propose a new type of embeddings for compounds that is a compositional model of the embeddings of individual components. Our experiments show that character-based embeddings are superior to word and compound embeddings in dependency parsing, and compound embeddings only outperform word embeddings when the part-of-speech (POS) information is unavailable. Thus, we conclude that it is the *morpho-syntactic* information of unknown compounds, not the semantic one, that is crucial for parsing German.

At the syntax level, we investigate challenges for *local* grammatical function labeler that are caused by *case syncretism*. In detail, we augment the grammatical function labeling component in a neural dependency parser that labels each head-dependent pair independently with a new labeler that includes a *decision history*, using Long Short-Term Memory networks (LSTMs). All our proposed models significantly outperformed the baseline on three languages: English, German and Czech. However, the impact of the new models is not the same for all languages: the improvement for English is smaller than for the non-configurational languages (German and Czech). Our analysis suggests that the success of the history-based models is not due to better handling of long dependencies but that they are better in dealing with the uncertainty in head direction.

We study the interaction of syntactic parsing with the semantic level via the problem of PP attachment disambiguation. Our motivation is to provide a *realistic* evaluation of the task where *gold* information is not available and compare the results of disambiguation systems against the output of a strong neural parser. To our best knowledge, this is the first time that PP attachment disambiguation is evaluated and compared against neural dependency parsing on predicted information. In addition, we present a novel approach for PP attachment disambiguation that uses biaffine attention and utilizes pre-trained contextualized word embeddings as semantic knowledge. Our end-to-end system outperformed the previous pipeline approach on German by a large margin simply by avoiding error propagation caused by predicted information. In the end, we show that parsing systems (with the same semantic knowledge) are in general superior to systems specialized for PP attachment disambiguation.

Lastly, we improve dependency parsing at the sentence level using *reranking* techniques. So far, previous work on neural reranking has been evaluated on English and Chinese only, both languages with a configurational word order and poor morphology. We re-assess the potential of successful neural reranking models from the literature on English and on two morphologically rich(er) languages, German and Czech. In addition, we introduce a new variation of a discriminative reranker based on graph convolutional networks (GCNs). Our proposed reranker not only outperforms previous models on English but is the only model that is able to improve results over the baselines on German and Czech. Our analysis points out that the failure is due to the lower quality of the k-best lists, where the gold tree ratio and the diversity of the list play an important role.

Zusammenfassung

Syntaktisches Parsen hat zum Ziel, die Struktur eines Satzes basierend auf einer vordefinierten formalen Grammatik zu analysieren. Es ist damit eine Schlüsselkomponente in vielen Pipelines für die Verarbeitung natürlicher Sprache und stellt wichtige Informationen für Anwendungen im Bereich des Verstehens natürlicher Sprache (Natural Language Understanding, NLU) bereit, wie z.B. die Extraktion von Informationen oder die Stimmungsanalyse.

Obwohl einige neuronale syntaktische Parser sehr hohe Präzisionswerte erzielen, fokussieren sich viele Untersuchungen auf nur wenige Sprachen (wie Englisch oder Chinesisch) oder studieren vorwiegend sprachunabhängige Konfigurationen. Daher fehlt es an Studien, die sich auf nur eine bestimmte Sprache konzentrieren und spezifische Analysestrategien für diese Sprache hinsichtlich ihrer sprachlichen Eigenschaften entwerfen.

Diese Arbeit stellt die deutsche Sprache ins Zentrum des Interesses und entwickelt akkuratere Methoden für die Analyse der besonderen syntaktischen Eigenschaften des Deutschen. Dazu werden neuartige Methoden aus dem Bereich des "Deep Learning" und der neuronalen Netze kombiniert mit Techniken, die die Herausforderungen an die automatische Analyse addressieren, die durch die sprachspezifischen Eigenschaften des Deutschen entstehen.

Im Vergleich zum Englischen besitzt das Deutsche eine sehr viel reichere Morphologie und zeichnet sich durch eine nicht-konfigurationelle Wortfolge und Kasus-Synkretismus aus. Es ist die Kombination dieser Eigenschaften, die das Parsen des Deutschen zu einer interessanten und herausfordernden Aufgabe macht.

Da syntaktisches Parsen sprachliches Wissen auf verschiedenen Ebenen des Sprachverständnisses erfordert, schlage ich vor, die Performanz von syntatischen Parsern auf jeder dieser Ebene zu studieren, um die Akkuratheit von syntaktischen Parsern fürs Deutsche insgesamt zu verbessern. Die dabei berücksichtigten Ebenen sind die (Sub-)Wortebene, die syntaktische Ebene, die semantische Ebene und die Satzebene.

Auf der (Sub-)Wortebene untersuchen wir den hohen Anteil an unbekannten (d.h., nicht in den Trainingsdaten vorhandenen) Worten in deutschen Datensets, bedingt durch die produktive Bildung von *Komposita* im Deutschen. Um dieses Problem zu bearbeiten, schlagen wir eine neue Art von Einbettungen (Embeddings) für Komposita vor, die sich aus den Einbettungen der einzelnen Wortkomponenten zusammensetzen. Unsere Experimente zeigen, dass zeichenbasierte Einbettungen den wortbasierten und den kompositionellen Einbettungen beim Dependenz-Parsen überlegen sind und Parsingergebnisse für zusammengesetzte Einbettungen die für Worteinbettungen nur dann übertreffen, wenn Wortarten-Informationen nicht verfügbar sind. Wir schließen daraus, dass es nicht die semantische, sondern die *morphosyntaktische* Information unbekannter Wortverbindungen ist, die für das Parsen des Deutschen entscheidend ist.

Auf der Syntaxebene untersuchen wir Herausforderungen für die *lokale* Bestimmung von grammatikalischen Funktionen aufgrund von *Kasus-Synkretismus*. Genauer gesagt, erweitern wir die Auszeichnunskomponente eines neuronalen Dependenzparsers, die jedem Kopf-Dependenten-Paar unabhängig von anderen Paaren eine grammatikalische Funktion zuweist. Die vorgeschlagene Erweiterung basiert auf Long Short-Term Memory Networks (LSTM) und berücksichtigt den bisherigen Entscheidungsverlauf (*decision history*). Die Ergebnisse unserer Modelle zeigen signifikante Verbesserungen für drei Sprachen: Englisch, Deutsch und Tschechisch.

Die Auswirkungen der neuen Modelle sind jedoch nicht für alle Sprachen gleich: Die Verbesserungen fürs Englische fallen geringer aus als die für nicht-konfigurationelle Sprachen (Deutsch und Tschechisch). Unsere Analyse legt nahe, dass der Erfolg der entscheidungsverlaufbasierten Modelle nicht auf einem besseren Umgang mit langen Abhängigkeiten beruht, sondern dass sie besser mit der Unsicherheit umgehen können, ob der Kopf des Dependenten rechts oder links vom Dependenten zu finden ist.

Auf der semantischen Ebene untersuchen wir das Problem der Disambiguierung von PP-Anhängungen, das für einen hohen Anteil an Fehlern beim syntaktischen Parsen verantwortlich ist. Unsere Motivation ist es, eine *realistische* Evaluation der Aufgabe zu liefern, bei der keine *Goldinformationen* verfügbar sind, und die Ergebnisse von Disambiguierungssystemen mit der Ausgabe eines starken neuronalen Parsers zu vergleichen. Nach unserem besten Wissen ist dies das erste Mal, dass Systeme zur Disambiguierung von PP-Anhängungen in einem realistischen Setting evaluiert und mit neuronalen Dependenzparsern verglichen werden.

Darüber hinaus präsentieren wir einen neuartigen Ansatz zur Disambiguierung von PP-Anhängungen, bei dem ein *biaffine attention*-Mechanismus genutzt wird und vortrainierte kontextualisierte Worteinbettungen als semantisches Wissen verwendet werden. Unser End-to-End-System übertrifft den bisherigen Pipeline-Ansatz fürs Deutsche um ein Vielfaches, indem es die durch vorhergesagte Informationen verursachte Fehlerfortpflanzung verhindert. Am Ende zeigen wir, dass Parsing-Systeme, die über das gleiche semantische Wissen verfügen, generell Systemen überlegen sind, die auf die Disambiguierung von PP-Anhängungen spezialisiert sind.

Zuletzt betrachten wir die Satzebene, wo wir das Parsen von Dependenzen mithilfe von *Reranking*-Techniken verbessern. Bisher wurden neuronale Rerankingsysteme nur auf englischen und chinesischen Daten evaluiert, beides Sprachen mit einer stark konfigurationellen Wortfolge und einer eher verarmten Morphologie. Wir präsentieren eine neue Bewertung des Potenzials erfolgreicher neuronaler Rankingmodelle aus der Literatur fürs Englische und für zwei morphologisch reich(er)e Sprachen, Deutsch und Tschechisch. Darüber hinaus führen wir eine neue Variante eines diskriminativen Rerankers ein, der auf Graph Convolutional Networks (GCNs) basiert. Unser vorgeschlagener Reranker übertrifft nicht nur frühere Modelle fürs Englische, sondern ist auch das einzige Modell, das in der Lage ist, die Ergebnisse gegenüber den Referenzwerten für Deutsche und Tschechische zu verbessern. Unsere Analyse zeigt, dass Rerankingfehler häufig auf die geringere Qualität der k-besten Liste zurückzuführen ist, bei denen der Anteil der Goldbäume in der Liste der k-besten Parsebäume sowie die Diversität der Liste eine wichtige Rolle spielen.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Humanity has long been dreaming about Artificial Intelligence (AI): machines or computers that have cognitive abilities similar to humans, such as thinking, learning, and decision-making. To facilitate human-machine communication, researchers work on Natural Language Understanding (NLU) components to equip intelligent agents with the ability to *understand natural languages*. Although we are still far from having real AI, NLU has gradually become a desirable feature in the design of modern devices. Today, voice assistants can be found in many households, not only for device control, but they can also provide us with information or other services, such as answering questions, ordering food, or booking a flight. While the progress made in this area seems astonishing, we still have to wonder *how much* human language these devices actually understand. Or, put differently, what does it *mean* to understand natural languages?

Let us consider the following example of a question that can be answered by many intelligent assistants:

Who's the current president of the United States of America?

To understand and answer this question, a voice assistant system must be able to process and combine information on many different levels, as outlined below:

- The system must be able to recognize that *States* is a plural noun, or *Who's* is the contraction of *Who is*. This is knowledge on the **morphological and lexical** level.
- The system also needs to understand the structural relationship between the words in the sentence. In this example, *the current president of the United States of*

America has a different meaning from *the United States of America of the current president*. This includes knowledge about **syntax**.

- On the next level, the system needs to understand the meaning of each word, and also needs to understand that the multi-word expression *the United States of America* refers to a country. This is **semantic** knowledge.
- The system also needs to correctly interpret the intention of the question, i.e., the speaker is expecting an answer rather than waiting for it to perform another action. The knowledge about the intention of the speaker is situated on the level of **pragmatics**.
- Finally, to be able to answer the question, the system has to interpret the meaning of the phrase *the current president* as the incumbent president at the time the speaker asks such question. This type of information is called **world knowledge**.

If a system can answer this question correctly, does it mean that the system understands natural languages on all levels? Unfortunately, it is harder to assess NLU than it seems. This is partly due to the fact that the latest state-of-the-art architectures for NLU tasks (question answering, information retrieval, etc.) are end-to-end neural models which makes it hard to understand what the system has learned and whether it was able to provide the correct answer for the right reasons. In fact, neural models have been shown to achieve high performance on benchmark data sets by learning *biased features*, i.e., surface patterns that are highly correlated to target labels (Gururangan et al., 2018; McCoy et al., 2019).

In this thesis, I argue that NLU should be *interpretable* and the success of a system should be demonstrated on *each level* of analysis. Therefore, I decided on **syntactic parsing** as a suitable task for studying the performance of Natural Language Processing systems on different levels of language understanding.

Why syntactic parsing? *Syntactic parsing* is the task of analyzing the structure of a sentence based on some predefined formal theory, thus assigning meaning to structural components. To be successful in syntactic parsing, a system (a *parser*) not only needs to be able to incorporate knowledge from the lower levels (morphology, lexicon, and syntax) but also interact with the semantic level. Different knowledge levels in syntactic parsing are introduced in section 1.2. Also, in contrast to NLU tasks, parsing systems cannot solely rely on surface heuristics to perform well on benchmark data sets.

Since 2015, breakthroughs in neural network research have pushed the state-of-the-art results for dependency parsing beyond 94% unlabeled accuracy for English, thus narrowing the gap to human performance on the same task. This raises the question of whether syntactic parsing is a solved task and if more research is needed in this area. The answer to this question is a clear No and there is still a lot to explore regarding neural syntactic parsing. First of all, the high parsing accuracy for English is only achieved when the parser is trained and tested on data from the same source. Also, benchmark data sets for syntactic parsing often include newswire texts. However, in practice, we cannot guarantee that the data of interest comes from the same domain as was used to train the parser. Often, we need to parse text from a different domain with many differences in word usage and structure complexity. In addition, syntactic parsers are often evaluated on a few prominent languages only (e.g., English, Chinese) or in language-agnostic settings (e.g., with Universal Dependencies treebanks). Thus, we still need more research that focuses on just one specific language and studies the linguistic phenomena of that language in more detail. This is also the reason for my choice of language, as described below.

Why syntactic parsing for German? The main objective of this thesis is to investigate syntactic parsing techniques for German. Despite being a high resource language, specific strategies for parsing German with regards to its linguistic properties are still underexplored.

Person	Singular	Plural
1st	schlaf**e**	schlaf**en**
2nd	schl**äfst**	schlaf**t**
3rd	schl**äft**	schlaf**en**

Table 1.1: The conjugation of the German verb *schlafen* (to sleep) in the present tense

	Masculine	Feminine	Neuter	Plural
Nominative	lecker**er**	lecker**e**	lecker**es**	lecker**e**
Accusative	lecker**en**	lecker**e**	lecker**es**	lecker**e**
Dative	lecker**em**	lecker**er**	lecker**em**	lecker**en**
Genitive	lecker**en**	leck**er**	lecker**en**	lecker**er**

Table 1.2: The strong inflection forms of the German adjective *lecker* (delicious)

Compared to English, German has a *richer morphology* that leads to many out-of-vocabulary words, causing data spareness. In particular, verb conjugation in German results in more distinctive forms, and German adjectives also inflect regarding the case and the gender of the noun they modify. Tables 1.1 and 1.2 illustrate examples of verb and adjective inflection in German. *Compounding* is another factor that increases the number of unknown words in the data. A *compound* is a word that consists of more than one stem. In German, the individual components are written without spaces, resulting in new word forms. For example, the compound *Wohnungsreinigung* (house cleaning) is made from two words: *Wohnung* (house) and *reinigung* (cleaning).

(1)

Es	*regnet*	stark	.	(verb-second)
It	rains	heavily	.	

"It rains heavily."

(2)

Ich	*bleibe*	zu	Hause	,	wenn	es	stark	*regnet*	.	(verb-second / verb-last)
I	stay	at	home	,	when	it	heavily	rains	.	

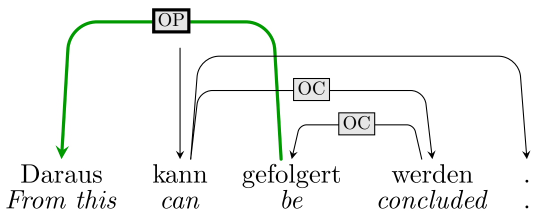
"I stay at home when it rains heavily."

(3)

Wenn	es	stark	*regnet*	,	*bleibe*	ich	zu	Hause	.	(verb-last / verb-first)
If	it	heavily	rains	,	stay	I	at	home	.	

"If it rains heavily, I stay at home."

Figure 1.1: Examples of verb positions in German sentences



"From this, it can be concluded."

Figure 1.2: A non-projective dependency tree of a German sentence from the SPMRL 2014 Shared Task data. The non-projective arc is marked with a thick line.

German has a *semi-free word order* that often causes non-continuous structures. While the position of German verbs in a sentence is very strict (they can either be at the first, second, or the last position of a sentence/phrase) (see figure 1.1), the ordering of other components is quite flexible. As a result, some syntactic trees are *non-projective*, i.e., they cannot be drawn without crossing edges on a plane, and parsing systems developed for English are usually not able to handle this issue. In the German TIGER corpus, a well-known benchmark corpus, about one-third of the trees are non-projective. Figure 1.2 displays the non-projective dependency tree of a German sentence.

(4)	Die Frau	jagt	die Ente	.
	The woman$_{\text{NOM/ACC}}$	chases	the duck$_{\text{NOM/ACC}}$	.

“The woman chases the duck.” / “The duck chases the woman.”

Figure 1.3: An example of case syncretism in German. Both *Die Frau* and *Die Ente* can be the subject in this sentence, resulting in two different readings.

Another characteristic that may affect parsing performance for German is *case syncretism*, which refers to the fact that different cases of a word may have the same surface form. In German, the nominative and accusative forms of feminine, neuter, and plural nouns are identical. This causes additional ambiguity that is hard to handle for a statistical parser. See figure 1.3 for an example of case syncretism in German.

To sum up, it is the combination of rich morphology, semi-free word order, and case syncretism that makes parsing German an interesting and challenging task. My hypothesis is that these language-specific characteristics should be taken into account in order to build a good parser for German. This forms the research methodology that I will describe in the next part.

Scope of the thesis In this work, I focus on *dependency parsing* as a more semantically transparent and flexible framework (in comparison to constituency parsing) to analyze the structure of German sentences. I take state-of-the-art models in dependency parsing, specifically *neural network models*, and experiment with them on German data to study the effect of language-specific properties on those language-agnostic approaches. The results then are analyzed to provide a deeper understanding of the technique, and from that, I develop new methods and more accurate models to improve parsing for German. Since my interest is to understand the effect of linguistic properties on the accuracy of models, my experiments often compare results obtained on German (with richer morphology, semi-free word order) with the ones on English, a language with impoverished morphology and configurational word order, and Czech, a morphologically-rich language with free word order.

1.2 Levels of Language Understanding in Syntactic Parsing

Syntactic parsing is a task that requires many levels of language understanding. Because of that, I propose to study and improve the knowledge of parsing models at each linguistic level in order to improve syntactic parsing for German. This is also the way the thesis is structured: each chapter of this work will consider a specific level of understanding in syntactic parsing. Specifically, these levels are:

- **Word level**: At the word level, German has more inflected forms compared to morphologically poorer languages (such as English), which leads to a high amount of words that have never been seen in the training data. This makes parsing German (or morphologically rich(er) languages in general) a challenging task. Another factor contributing to this challenge is *compounding*. In German, word components can be merged into new word forms (without spaces), and since compounding is highly productive, it is a major source for unseen words. In chapter 4, I will look at different techniques and their efficiency to handle unknown words in parsing German, including a new *subword embedding* type based on compounds.

- **Syntactic level**: In dependency parsing, an (directed) *edge* connecting two words in a sentence denotes a relation between them. The *label* of the edge represents the type of relation, or the *grammatical function*. For example, in figure 1.2, the edge from *gefolgert* to *Daraus* has the label *OP*, meaning *Daraus* is the prepositional object of the verb *gefolgert*. The identification of the grammatical function of each word is essential for interpreting the meaning of a sentence, especially for languages with a non-configurational word order. In German, labeling grammatical functions is a very challenging task because of *case syncretism*, i.e., when different cases correspond to the same inflected form. Recent neural dependency parsers assign grammatical function labels for each edge independently, leading to well-known errors of local parsers such as duplicate subjects for the same predicate. A simple way to avoid these errors is to make the parser aware of the *context* while assigning those labels. Thus, in chapter 5, I propose to improve grammatical function labeling by augmenting the labeling component of a parser with a *decision history*.

- **Semantic level**: Prepositional phrase (PP) attachment disambiguation, the task of identifying the correct attachment site for each preposition in the syntax tree, has been identified as one of the major sources for parser errors. Although it can be considered as a subtask in syntactic parsing, morpho-syntactic information is often insufficient to resolve the ambiguity in PP attachment, and additional *semantic* information or even *world knowledge* is needed. In chapter 6, I will revisit the PP attachment disambiguation problem with a case study on German and present models that significantly outperform the current state-of-the-art.

- **Sentence level**: Despite their impressive results, most state-of-the-art parsers are *local* and *greedy* and are thus expected to have problems finding the best *global* parse tree. *Reranking* is a technique to improve parsing performance on the output of a base parser by adding a global and complete view of the tree in contrast to the local and incomplete features used in local parsing. In chapter 7, I will re-evaluate the potential of existing reranking models on three languages: English, German and Czech, and analyze the difference in their performance between the three languages.

1.3 Thesis Outline & Contributions

Based on the structure of different linguistic levels described in the previous section, this thesis is organized as follows. The next two chapters provide the background knowledge and references related to the work in this thesis: Chapter 2 introduces the concepts and techniques of neural network models that are used in natural language processing. In chapter 3, we first formally define and give an overview of conventional approaches for *dependency parsing*. We then discuss the development and current trends in neural dependency parsing in the second half of the chapter.

In chapter 4, we study the challenges for parsing German at the lexicon level posed by *unknown words*, especially those caused by *compounds*. We hypothesize that the meaning of a compound can be inferred from the meaning of its components. Therefore, we propose a *new* type of subword embeddings, called *compound embeddings*, which is a compositional model of the embeddings of individual components. We then compare the effect of compound embeddings and character-based word embeddings on parsing German when POS tag information is present, and when it is absent. Our experiments with the new type of embeddings show that compound embeddings only outperform word embeddings in dependency parsing when the part-of-speech (POS) information is absent, and character-based embeddings always perform better than both of them. Thus, we conclude that it is not the *semantic* information of the unknown compounds that is crucial for parsing German, but their *morpho-syntactic* information.

In chapter 5, we alter the grammatical function labeling component of a parser to improve parsing at the structural and word order level. Specifically, we replace the labeler in a neural dependency parser that labels each head-dependent pair independently with the new labeler that includes a *decision history*. Following previous work, we model grammatical function labeling as a sequence labeling task and assign labels to each edge, using Long Short-Term Memory networks (LSTMs). We experiment with three different input orders: (1) *surface ordering*, where tree nodes are ordered according to their surface order in the sentence; (2) *BFS ordering*, where tree nodes are ordered according to a breadth-first traversal (BFS) of the tree; and (3) *tree ordering*, where nodes are labeled based on the top-down order from the root node in the unlabeled tree. We report the results on four data sets for three different languages: English, German and Czech. All our proposed models significantly outperform the baseline on three languages, but the best improvement on core argument functions is achieved with the BFS ordering.

In chapter 6, our goal is to provide a *realistic evaluation* of specialized systems for PP attachment disambiguation (that additionally requires semantic information), with a case study on German. In particular, we re-evaluate previous approaches to PP attachment disambiguation for German in a real-world scenario where *gold* information is not available and compare the results against the output of a strong neural parser. To our best knowledge, this is the first time that PP attachment disambiguation is evaluated and compared against neural dependency parsing on predicted information. We also propose a new model for PP attachment disambiguation with biaffine attention that takes into account all words in the sentence as candidates without the need to restrict the input candidate heads, as has been done in previous approaches. In settings with only *predicted* information, our new model outperforms recent work on German by a large margin, but its performance is still worse than that of a strong neural parser. Our experiments suggest that parsing systems are in general superior to systems specialized for PP attachment disambiguation.

In chapter 7, we revisit *reranking* techniques to improve dependency parsing at the sentence level. So far, previous work on neural reranking has been evaluated on English and Chinese only, two languages with a configurational word order and poor morphology. We bridge this gap by re-assessing the efficiency of those systems not only on English but also on two additional morphologically rich(er) languages: German and Czech. In addition to evaluating two successful rerankers from the literature, we introduce a new type of discriminative reranker based on graph convolutional networks (GCNs). Finally, we analyze the differences in performance between the three reranking models on three languages. Our proposed reranker with GCNs not only outperforms all previous models on English but is the only model that is able to improve results over the baselines on German and Czech.

We summarize our findings and contributions of this thesis, as well as discuss potential directions for future work in chapter 8.

1.4 Published Work

The research presented in this thesis is an extension of the published work by the author of this thesis. We list these publications below.

1. Bich-Ngoc Do et al. (2017). "What Do We Need to Know about an Unknown Word When Parsing German". In: *Proceedings of the First Workshop on Subword and Character Level Models in NLP*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 117–123. DOI: 10.18653/v1/W17-4117

2. Bich-Ngoc Do and Ines Rehbein (2017). "Evaluating LSTM Models for Grammatical Function Labelling". In: *Proceedings of the 15th International Conference on Parsing Technologies*. Pisa, Italy: Association for Computational Linguistics, pp. 128–133

3. Bich-Ngoc Do and Ines Rehbein (2020a). "Neural Reranking for Dependency Parsing: An Evaluation". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, pp. 4123–4133. DOI: 10.18653/v1/2020.acl-main.379

4. Bich-Ngoc Do and Ines Rehbein (2020b). "Parsers Know Best: German PP Attachment Revisited". In: *Proceedings of the 28th International Conference on Computational Linguistics*. Barcelona, Spain (Online): International Committee on Computational Linguistics, pp. 2049–2061. DOI: 10.18653/v1/2020.coling-main.185

Chapters 4 and 5 have been published as Do et al. (2017) and Do and Rehbein (2017) respectively. Chapters 6 and 7 are extensions of Do and Rehbein (2020b) and Do and Rehbein (2020a).

CHAPTER 2

Background

In this chapter, we introduce fundamental concepts (section 2.1) and techniques (section 2.2) of neural network models. The concepts and techniques chosen to be presented here provide the basis for understanding the work in subsequent chapters of this thesis. Since neural networks have become an enormous field, this chapter cannot cover every topic in detail, but rather serves as a reference to previous work. We refer the readers to Goldberg (2017) and Goodfellow et al. (2016) for an in-depth introduction of these topics.

2.1 Artificial Neural Networks at a Glance

Artificial neural networks (ANNs), or neural networks for short, have initially been proposed as computational models of the human brain (McCulloch and Pitts, 1943). Today, the models only vaguely resemble the biological brains in the sense that they are composed of many interconnected computational nodes similar to *neurons*. With recent breakthroughs in computation hardware and training techniques, researchers can now efficiently train large neural networks with many layers. Because of that, the term *deep learning* is sometimes used synonymously with neural network techniques, although its meaning is not necessarily restricted to neural models. Neural networks have become the most powerful machine learning models of this decade, achieving state-of-the-art results in many fields, including natural language processing (NLP), speech processing, and computer vision.

2.1.1 Perceptrons

A *neuron* in the nervous system consists of a *soma*, the cell body, and an *axon*. An axon branches at one end and connects to the somas of other neurons. Via axon connections,

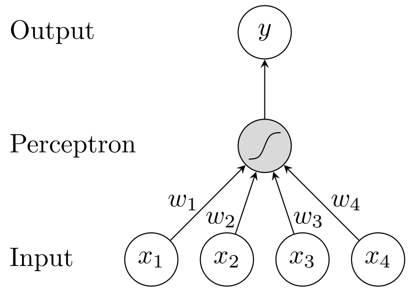


Figure 2.1: A perceptron with 4 inputs. Here the bias term is omitted.

electrical and chemical signals are transmitted through the net of neurons, but a neuron only releases an output signal if the strength of the input excitation exceeds a certain *threshold*. The basic unit of ANNs is a *perceptron*, a computational model inspired by the neuron. In a similar fashion, a perceptron (figure 2.1) combines its scalar inputs, passes the combined result through a non-linear function, and outputs a scalar number. The most common way to combine inputs is by using linear functions. Let $x_1, x_2, ..., x_n$ be the inputs to a perceptron. Its output y is then calculated as:

$$y = \varphi\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.1}$$

where $w_1, w_2, ..., w_n$ are the *weights*, b is the *bias*, and $\varphi(\cdot)$ is a nonlinear function called the *activation function*.

In mathematical notation, a perceptron is a function $f(\mathbf{x}, \theta)$ that maps an input vector x to a scalar output y with parameters θ:

$$y = f(\mathbf{x}; \theta) = \varphi\left(\mathbf{x}^\top \mathbf{w} + b\right) \tag{2.2}$$

where $\mathbf{x} \in \mathbb{R}^m,\ \mathbf{w} \in \mathbb{R}^m,\ b \in \mathbb{R}$.

2.1.2 Feed-forward Neural Networks

By using more than one perceptron, we can create a mapping between an input vector x and an output vector y of arbitrary dimensions:

$$\mathbf{y} = f(\mathbf{x}, \theta) = \varphi\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right) \tag{2.3}$$

where $\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n,\ \mathbf{W} \in \mathbb{R}^{n \times m},\ b \in \mathbb{R}^n$. These perceptrons together are called a *layer* of perceptrons. By combining multiple layers of perceptrons, we can form a *multilayer perceptron* (MLP), where computation between layers is carried out in one direction (see figure 2.2). Thus, MLPs are also known as *feed-forward* neural networks. The input to an MLP is called the *input layer*, and the outer-most nonlinear

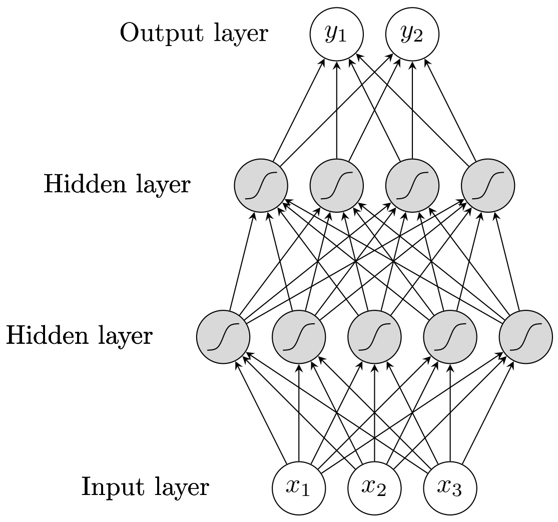


Figure 2.2: A multilayer perceptron with 2 hidden layers

transformation is called the *output layer*. Other perceptron layers are called *hidden layers* because their outputs are only used within the MLP. Consider the MLP in figure 2.2 with two hidden layers. The outputs of the two hidden layers are:

$$\mathbf{h}_1 = f^{(1)}(\mathbf{x};\theta_1) = \varphi_1\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1\right) \tag{2.4}$$

$$\mathbf{h}_2 = f^{(2)}(\mathbf{x};\theta_2) = \varphi_2\left(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2\right) \tag{2.5}$$

where $\theta_k = \{\mathbf{W}_k, \mathbf{b}_k\}$ are parameters of the k-th layer. The outputs of the MLP are:

$$\mathbf{y} = f^{(3)}(\mathbf{x};\theta_3) = \varphi_3\left(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3\right) \tag{2.6}$$

Thus, an MLP also defines a mapping:

$$\mathbf{y} = f(\mathbf{x};\theta) = f^{(n)}\left(\cdots f^{(2)}\left(f^{(1)}\left(\mathbf{x};\theta_1\right);\theta_2\right)\cdots;\theta_n\right) \tag{2.7}$$

The number of layers in an MLP is called the *depth* of the model, while the number of perceptrons in each hidden layer, or the layer *size*, determines the *width* of the model.

The role of nonlinearity in the activation function is crucial, as it gives MLPs the ability to approximate nonlinear functions. Without it, the whole MLP becomes just a linear combination of its inputs. The *sigmoid* function was the earliest choice of activation function for neural networks but has been discouraged because it makes neural networks hard to train[1]. The *tanh* function is an alternative for the sigmoid, but perhaps the most popular activation function is the *rectifier* function (Glorot et al., 2011), also known as rectified linear unit (ReLU) due to its simplicity and effectiveness in many systems:

$$\mathrm{ReLU}(x) = \max(0, x) \tag{2.8}$$

[1]The gradient of the sigmoid function saturates to zero at its two ends, and as a result, the network stops learning and the parameters of the neural network remain unchanged during training.

Optimization MLPs and neural networks, in general, are trained with iterative, mini-batch *gradient-based* methods to minimize a predefined *cost function* since its cost is nonconvex due to nonlinearity. The training process involves three steps. For each batch of inputs:

- *Forward* propagation: The values of each node in the model are computed with respect to each input in the batch.
- *Backward* propagation: The gradients of each node are computed for each input in the batch.
- *Update*: The parameters are updated based on the aggregation of gradients in the whole batch.

The process continues until training converges, or some predetermined early stopping criteria are met. A gradient-based optimization algorithm defines specific rules to adjust the model parameters in each step. The term *backpropagation*, or *backprop* for short, is sometimes inaccurately used to refer to the gradient-based algorithms used to train neural network models, although it only refers to the method for computing the gradients as in the backward propagation step. Also, back-propagation and gradient-based methods are not only limited to MLPs or neural networks but can be applied to any function in general.

Capability The *universal approximation theorem* (Cybenko, 1989; Hornik, 1991) states that an MLP with one hidden layer is a *universal approximator*, i.e., it can approximate äll continuous functions on a closed and bounded subset of $\mathbb{R}^n$, and any function mapping from any finite dimensional discrete space to another"(Goldberg, 2017) with an arbitrary accuracy. The theorem might be misunderstood as meaning that an MLP with one hidden layer is sufficient to approximate any continuous function and it is thus not necessary to go beyond one-hidden layer MLP to deep and more complex neural network structures. However, what it really means is that there will be a large, shallow MLP that is able to *represent* any function of interest (*capability*), but it does not guarantee that the training algorithm will be able to *find* such a model (*trainability*). In practice, *trainability* is a more important factor than capability, determining the effectiveness of neural networks.

2.1.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a family of neural networks that processes sequential data. For example, many problems in NLP deal with sequences as input,

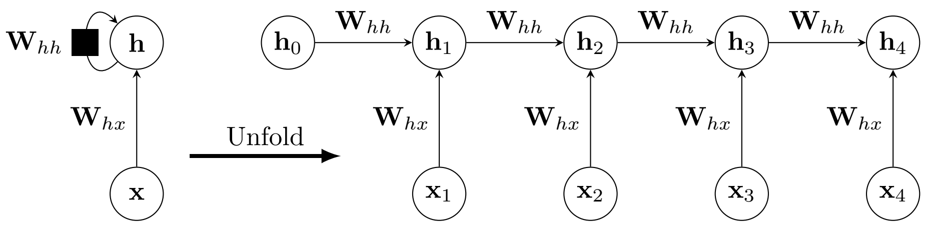


Figure 2.3: A vanilla RNN (left) and its unfolded computational graph (right) when processing a sequence of length 4. The black square indicates a delay of one time step.

like words (sequences of characters), sentences (sequences of words), and documents. In speech processing, the input is a sequence of acoustic vectors sampled from an analog sound wave. It is thus important to treat the input sequence as a whole rather than to consider each element in the sequence independently.

While the sizes of an MLP are determined before training, input sequences often vary in length. In order to process arbitrary length sequences with MLPs, the sequence must be converted to a fixed-size vector. This can be done by processing the sequence in a fixed-size window (like n-grams), or by aggregation (like summing or averaging). Neither technique is ideal for capturing dependencies in sequences: either the long-term dependencies are sacrificed, or the order of the input sequence is ignored. Unlike MLPs where information flows in only one direction, RNNs allow *feedback* connections, where the output of the model is fed back to itself in the next step. In general, an RNN maps a *sequence* of input $\mathbf{x}_{1:n} = \mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$ to some output $\mathbf{y}$:

$$\mathbf{y} = f_{\text{RNN}}(\mathbf{x}_{1:n}; \theta) \tag{2.9}$$

An RNN is often used as a component in neural network models rather than an independent model, and whether $\mathbf{y}$ takes the form of a fixed-size vector or a sequence of vectors depends on its usage. The *inner function* of an RNN is a recursive function:

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}; \theta) \tag{2.10}$$

The index t is usually referred to as the *time step*. $\mathbf{h}_t$ is the *hidden state* of the RNN at time step t. The equation of the inner function shows that RNNs also take into consideration the output $\mathbf{h}_{t-1}$ of the previous time step, in addition to the input $\mathbf{x}_t$ of the current time step.[2] The earliest and simplest form of RNNs, also known as *vanilla* RNNs (figure 2.3), are proposed by Elman (1990):

$$\mathbf{h}_t = f_{\text{vanilla}}(\mathbf{x}_t, \mathbf{h}_{t-1}; \theta) = \tanh(\mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}) \tag{2.11}$$

[2]Although feedback connections of RNNs can also be from later states rather than hidden states, here we only focus on the most popular architecture of RNNs.

where $\mathbf{x}_t \in \mathbb{R}^m$, $\mathbf{h}_t \in \mathbb{R}^n$, $\mathbf{W}_{hx} \in \mathbb{R}^{n \times m}$, $\mathbf{W}_{hh} \in \mathbb{R}^{n \times n}$, $\mathbf{b} \in \mathbb{R}^n$. The mathematical formula of vanilla RNNs can also be expressed as:

$$\mathbf{h}_t = \tanh\left(\mathbf{W}\begin{bmatrix}\mathbf{x}_t \\ \mathbf{h}_{t-1}\end{bmatrix} + \mathbf{b}\right) \tag{2.12}$$

where $\mathbf{W} \in \mathbb{R}^{n \times (m+n)}$. With a finite number of time steps τ, the computation of $\mathbf{h}_\tau$ can be *unfolded* into an equation of previous time steps $\mathbf{h}_1, ..., \mathbf{h}_{\tau-1}$. For example, the hidden state of a vanilla RNN at time step 3 (see figure 2.3) is:

$$\mathbf{h}_3 = f_{\text{vanilla}}\left(\mathbf{x}_3, f_{\text{vanilla}}\left(\mathbf{x}_2, f_{\text{vanilla}}\left(\mathbf{x}_1, \mathbf{h}_0; \theta\right); \theta\right); \theta\right) \tag{2.13}$$

$\mathbf{h}_0$ is also a parameter of the RNN and is randomly generated or initialized as zeros. In its unfolded form, the formula of RNNs looks very much like that of a deep MLP where all layers share the same set of parameters.

Bidirectional RNNs The hidden state $\mathbf{h}_t$ is conditioned on inputs $\mathbf{x}_1, ..., \mathbf{x}_t$, so in theory, it captures the input information from the beginning up to time step t. As before, $\mathbf{h}_T$ encodes information of the whole input sequence. Thus, $\mathbf{h}_T$ is usually used as the learned representation of the input sequence, as needed in sentence classification tasks, or as a summary of a sequential structure like stacks (see section 3.2.3). In a similar fashion, a hidden state of an RNN can also be conditioned on its *future* rather than its *past* by simply reversing the processing direction of the RNN:

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t+1}, \theta) \tag{2.14}$$

The left-to-right RNNs are usually called *forward* RNNs, and the right to left RNNs are called *backward RNNs*. A *bidirectional* RNN consists of one forward and one backward RNN, and returns two hidden states conditioned on the past and the future of the input sequence:

$$\begin{aligned}\mathbf{h}_t^F &= \text{RNN}^F(\mathbf{x}_t, \mathbf{h}_{t-1}^F) \\ \mathbf{h}_t^B &= \text{RNN}^B(\mathbf{x}_t, \mathbf{h}_{t+1}^B)\end{aligned} \tag{2.15}$$

Multilayer RNNs RNNs and bidirectional RNNs can also be stacked, like MLPs. To do so, the hidden states of the previous RNN become the input to the next RNN layer. With bidirectional RNNs, the forward and backward hidden states are often concatenated and passed to the next layer. In practice, two or more layers of RNNs work better than one layer, as appear to capture more abstract information that cannot be learned on the lower layers.

Vanishing and exploding gradients To train an RNN, its computation graph must be unfolded, and back-propagation is applied to compute the gradients for each node on the unfolded graph. This process is known as *backpropagation through time* (BPTT). BPTT is not only expensive, but the long computation chain causes well-known problems in deep neural networks and RNNs: the *vanishing* and *exploding* gradient problem. Because the gradients are computed through *too* many steps when the input sequence is long, the gradients either become smaller and smaller, then saturate to zero (*vanishing*), or they are accumulated and grow too large (*exploding*). When either of those happens, RNNs, especially vanilla RNNs, are difficult to train. Too small gradients inhibit the learning process, while too large gradients make training unstable. Several techniques are introduced to overcome these problems, including weight initialization strategies, batch normalization (Ioffe and Szegedy, 2015), and gradient clipping (Pascanu et al., 2013). Alternatively, RNNs with *gated architectures* are exclusively designed to deal with vanishing and exploding gradient problems.

Gated architectures Gated RNNs such as Long Short-Term Memory networks (LSTMs) (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Units (GRUs) (Cho et al., 2014) contain controlled paths that allow gradients to flow for a long time. Here, information flow is decided by *gates*, which are vectors of real values in the range of $(0, 1)$. They control how much information is kept or discarded. Gradients are stabilized with gate-controlled *memory states*, which have the basic form of:

$$\mathbf{s}_t = \mathbf{g}_t \odot \mathbf{x}_t + (1 - \mathbf{g}_t) \odot \mathbf{s}_{t-1} \tag{2.16}$$

where $\mathbf{u} \odot \mathbf{v}$ denotes the element-wise (Hadamard) product of two vectors $\mathbf{u}$ and $\mathbf{v}$. The formula of $\mathbf{s}_t$ contains two terms of which gradients are unlikely to vanish at the same time, thus gradients are stable even in long computation chains. Different architectures of gated RNNs define different gating mechanisms. For instance, LSTMs are formulated as follow:

$$\begin{bmatrix}\mathbf{i}_t\\ \mathbf{f}_t\\ \mathbf{o}_t\\ \tilde{\mathbf{c}}_t\end{bmatrix} = \begin{bmatrix}\sigma\\ \sigma\\ \sigma\\ \tanh\end{bmatrix}\left(\begin{bmatrix}\mathbf{W}_i\\ \mathbf{W}_f\\ \mathbf{W}_o\\ \mathbf{W}_c\end{bmatrix}\begin{bmatrix}\mathbf{x}_t\\ \mathbf{h}_{t-1}\end{bmatrix} + \begin{bmatrix}\mathbf{b}_i\\ \mathbf{b}_f\\ \mathbf{b}_o\\ \mathbf{b}_c\end{bmatrix}\right) \tag{2.17}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \tilde{\mathbf{c}}_{t-1} + \mathbf{i}_t \odot \mathbf{c}_t \tag{2.18}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \tag{2.19}$$

where $\mathbf{x}_t \in \mathbb{R}^m,\ \mathbf{h}_t \in \mathbb{R}^n,\ \mathbf{W}_i, \mathbf{W}_f, \mathbf{W}_o, \mathbf{W}_c \in \mathbb{R}^{n\times(m+n)},\ \mathbf{b}_i,\ \mathbf{b}_f,\ \mathbf{b}_o,\ \mathbf{b}_c \in \mathbb{R}^n$. σ is the sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.20}$$

Memory states in LSTMs are split into hidden states $\mathbf{h}_t$ as in vanilla RNNs, and cell states $\mathbf{c}_t$ that preserve memory. Although other gated RNNs have less complicated structures and achieve better results in certain cases, LSTMs are unarguably the most versatile architecture and became the default choice of recurrent structures. Keep in mind that when the effectiveness of different RNNs (or neural models in general) is compared, what matters is the *trainability* rather than *capacity*, as Collins et al. (2017) show that all common RNN models have nearly the same capacity, and vanilla RNNs have even slightly higher capacity but are harder to train.

2.1.4 Recursive Neural Networks

Recursive neural networks are a generalization of RNNs applied to tree structures. In the earliest works, recursive neural networks are applied to bottom-up binary trees (Socher et al., 2010; Socher et al., 2013). Later, vanilla RNNs in recursive neural networks are replaced with LSTM cells, and recursive RNNs are extended to work on trees with an arbitrary branching factor (Tai et al., 2015; Zhu et al., 2015; Kiperwasser and Goldberg, 2016a) as well as top-down trees (Le and Zuidema, 2014; Zhang et al., 2016). Figure 2.4 shows an example of vanilla recursive neural networks on a binary tree. The hidden states $\mathbf{p}_1$ and $\mathbf{p}_2$ are computed as:

$$\mathbf{p}_1 = \varphi(\mathbf{W}_1\mathbf{y} + \mathbf{W}_2\mathbf{z} + \mathbf{b}) \tag{2.21}$$

$$\mathbf{p}_2 = \varphi(\mathbf{W}_1\mathbf{x} + \mathbf{W}_2\mathbf{p}_1 + \mathbf{b}) \tag{2.22}$$

where $\mathbf{x},\ \mathbf{y},\ \mathbf{z},\ \mathbf{p}_1,\ \mathbf{p}_2 \in \mathbb{R}^n,\ \mathbf{b} \in \mathbb{R}^n,\ \mathbf{W}_1,\ \mathbf{W}_2 \in \mathbb{R}^{n\times n}$. On a sequence of length T, the computation graph of a recursive neural network is shorter than that of an RNN ($O(\log T)$ vs. $O(T)$). Li et al. (2015) show that recursive neural networks have an advantage over RNNs on tasks where long dependencies are crucial, although an approximation of recursive neural networks with RNNs can achieve comparable performance on these tasks. We use recursive neural networks in our work on grammatical function labeling (chapter 5) where we predict the label of a word based on its order on the tree, and also as a scoring model for tree reranking (chapter 7).

2.1.5 Attention Mechanism

In section 2.1.3, we have shown that the last hidden state $\mathbf{h}_T$ returned by RNNs encodes information of the whole input sequence. Thus, $\mathbf{h}_T$ can be used as a representation of the whole input sequence, e.g., if the input is a sequence of words, then $\mathbf{h}_T$ can be used as a representation of the sentence. Here, the whole sequence is compressed into a vector of size d. However, this becomes problematic when the

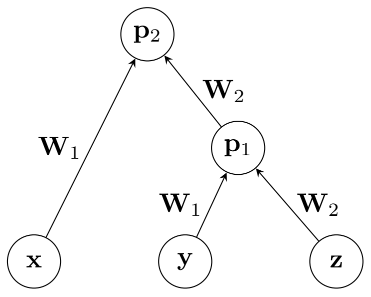


Figure 2.4: A recursive neural network on a binary tree

input sequence is very long, as more information needs to be compressed into a fixed-size vector. Because of that, the performance for systems that utilize the last hidden state degrades quickly as the sequence length increases.

To address this problem, Bahdanau et al. (2015) propose a technique called *attention mechanism*. The intuition behind this idea is to compute the sequence representation as the weighted sum of all hidden states, where the weights are *dynamically* determined based on the current situation. Given a sequence $\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_T$, the attention mechanism combines all hidden states to calculate the sequence representation $\mathbf{c}$:

$$\mathbf{c} = \sum_{t=1}^{T} \alpha_t \mathbf{h}_t \tag{2.23}$$

where α_t is a weight deciding the importance of the time step t to the final representation. α_t is not fixed, but adaptive:

$$\alpha_t = \frac{\exp(a(\mathbf{q}, \mathbf{h}_t))}{\sum_{t'=1}^{T} \exp(a(\mathbf{q}, \mathbf{h}_{t'}))} \tag{2.24}$$

Here, $a(\cdot)$ is the *alignment function* that calculates the *compatibility* of the hidden state $\mathbf{h}_t$ for a *query* vector $\mathbf{q}$. For example, in machine translation, $\mathbf{q}$ is the last hidden state of the RNN decoder (Bahdanau et al., 2015). Function $a(\cdot)$ can take many forms, as simple as the dot product between two vectors, or more complex forms like an MLP with one hidden layer.

Vaswani et al. (2017) generalize the attention mechanism with *scaled dot-product attention*. Inputs to the attention function are: a query matrix $\mathbf{Q}$ of size $m \times d_k$, a key matrix $\mathbf{K}$ of size $n \times d_k$, and a value matrix $\mathbf{V}$ of size $n \times d_v$. The output matrix is computed as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V} \tag{2.25}$$

The attention mechanism of Bahdanau et al. (2015) has $\mathbf{Q} = \mathbf{q}^\top$ and $\mathbf{K} = \mathbf{V} = [\mathbf{h}_1; ...; \mathbf{h}_T]^\top$.

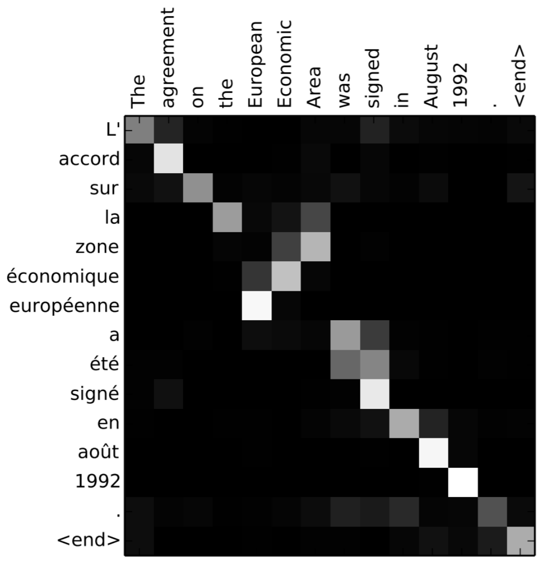


Figure 2.5: An example of attention visualization in machine translation (reproduced from Bahdanau et al. (2015)). The x-axis and y-axis correspond to the words in the source sentence (English) and the generated target sentence (French) respectively. Each pixel shows the attention weight α_{ij} of the j-th word in the source sentence for the i-th word in the target sentence, in gray scale (0: black, 1: white).

The attention mechanism provides an intuitive way to understand the model's decisions. The attention weights α indicate which positions in the input sequence are considered when generating a target word of the output sequence. Figure 2.5 illustrates the attention weights when translating a sentence from English to French. While word alignments are monotonic in most cases (indicated by the diagonal weights), there are some non-monotonic alignments. Despite the order of nouns and adjectives is different between English and French, the phrase *the European Economic Area* is correctly translated into *la zone économique europée ne*, and attention visualization shows that the model correctly aligns the source and the target words.

2.1.6 Transformer

The *Transformer* (Vaswani et al., 2017) is an encoder-decoder model for sequence-to-sequence tasks, such as machine translation. Before that, successful neural sequence-to-sequence models have employed RNN architectures with attention (section 2.1.5) (Bahdanau et al., 2015). RNN-based sequence-to-sequence models consist of a multilayer bidirectional LSTM as an *encoder* to read information from the input sequence, and a multilayer LSTM *decoder* to generate the output sequence step-by-step, that with the help of attention mechanism takes into account information from both the input sequence and the previously generated words.

The Transformer model completely eliminates the RNNs from the model architecture. Instead, its basic building block is the *multi-head self-attention* mechanism. Self-attention is basically the scaled dot-product attention introduced in section 2.1.5 where queries, keys, and values come from the same place. In this case, they are all computed based on the output sequence of the previous layer. Using the attention function multiple times results in multiple attention *heads*, which allows the model to jointly attend to information at different positions. The self-attention mechanism combines information from the left and right contexts, similar to bidirectional RNNs, but it disregards the input order when calculating the attentions. To make up for that, *positional encoding* is used to inject position information directly into the input sequence.

Both the encoder and decoder of the Transformer are a stack of identical blocks, where each block contains a multi-head self-attention layer and a fully connected feed-forward neural network. The multi-head self-attention layer in the decoder is masked to prevent it from attending to future information.

At a higher level, the Transformer and sequence-to-sequence models with RNNs function very much alike. However, self-attention is much faster than the recurrent operation that allows the model to grow deeper. Since its introduction, the Transformer model has been applied to many NLP tasks and achieved state-of-the-art results in most of them.

2.2 Neural Techniques for Natural Language Processing

From the previous sections, we are now equipped with the most important concepts of neural network models, which provide us with a powerful tool for learning any problem of interest. Up to this point, the input to neural networks are vectors of real numbers, and they are assumed to be predefined. This section deals with how to use neural network techniques with natural language data, where the majority of input features are discrete, categorical features, such as word forms, lemmas, part-of-speech (POS) tags, or characters.

2.2.1 Feature Representations: Embeddings

One-hot encoding In conventional machine learning models, categorical features are encoded as *one-hot vectors*. That is, a feature with k possible values is represented as a k-dimensional vector where only one dimension (corresponding to the current value) has value 1 and the rest are 0s. For example, consider the input sentence *The* *pig is flying* for which we want to encode the feature *POS tag of the first token*, or tag_1, that has 6 possible values: NOUN, VERB, ADJ, ADV, DET, and PREP. The one-hot encoding of the feature tag_1 for the sentence above is then:

$$\mathbf{f}(\text{tag}_1 = \text{DET}) = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}^\top \tag{2.26}$$

where the 5th dimension corresponds to the POS tag DET. A single (categorical) feature is also known as an *atomic* or a *core* feature. Many traditional linear models require core features to be combined manually, resulting in *combined features*. For example, given a feature *word form of the first token*, or word_1, which can take 100 values. Then, the combined feature word_1 & tag_1 is represented as a one-hot vector with $100 \times 6 = 600$ dimensions. Because of that, feature combination can lead to an explosion in the number of features, i.e., it represents a very sparse feature with a large number of dimensions. Concatenating the representations of all atomic and combined features gives us the input vector with n non-zero entries corresponding to the number of distinct features. Thus, using one-hot encoding results in *sparse* and *discrete* input vectors.

Embeddings In contrast to the sparse and discrete one-hot vectors, neural network models often represent input features as *dense* and *continuous* vectors of real values. Here, each core feature with k possible values is *embedded* in a d-dimensional space, producing a matrix $\mathbf{E}$ of size $d \times k$ where each column of $\mathbf{E}$ stands for one particular feature value. Usually, the dimension d is much smaller than the number of values k. Thus, the input dimension for using feature embeddings is much smaller, compared to one-hot encoding. The matrix $\mathbf{E}$ is called *embeddings*. It is part of the neural network model's parameters and thus is updated during model training. The initial values for the embeddings can be set to random numbers, or be initialized with an embedding matrix that has been learned in a different task. Here, the embeddings are said to be *pre-trained*, which will be discussed in more detail in section 2.2.2. With multilayer architectures, core features are automatically combined in the higher layers of neural network models, thus making the time-consuming process of feature engineering superfluous.

One-hot encoding vs. embeddings Representing arbitrary features with embeddings was pioneered by Collobert and Weston (2008), Collobert et al. (2011) and Chen and Manning (2014). Although almost all neural systems employ embeddings instead of one-hot vectors, both types of encoding can be used as input to neural networks. The most apparent advantage of embeddings over one-hot vectors is the smaller input dimension, which is more efficient in system implementation. However, the advantage is negligible when a feature only takes a small number of possible values. In fact, the difference between embeddings and one-hot encoding usage in neural networks is quite subtle. Using one-hot encoding as input makes the first layer of the model learn the feature embeddings (with some minor differences) (Goldberg, 2017). Still, it is worth noting that embeddings allow effortless *parameter sharing*, for instance, the features word$_1$ and word$_2$ can use the same embedding matrix, while this is harder to do with one-hot encoding.

The most important difference between one-hot encoding and embeddings, however, concerns their representation power. For one-hot encoding, the feature vectors for two different values are always independent, as the cosine between them is always 0. Because of that, the feature vector for word=*pig* is as dissimilar to the feature vector for word=*cat* as it is to the feature vector for word=*revolution*. For embeddings, however, the dense representation makes it possible to capture the *similarity* between two entities. If we believe that the word *pig* should be more similar to *cat* than to *revolution*, these vectors can be chosen such as:

$$\cos(\mathbf{v}_{\text{pig}}, \mathbf{v}_{\text{cat}}) < \cos(\mathbf{v}_{\text{pig}}, \mathbf{v}_{\text{revolution}}) \tag{2.27}$$

Therefore, it is the use of *pre-trained* embeddings that makes dense feature representations so powerful. In the next section, we will learn how to obtain good pre-trained representations for words.

2.2.2 Pre-trained Word Embeddings

Neural network models are powerful and allow us to learn good feature representations, given that enough training data is available. First, the embedding matrix is initialized randomly, then it is trained the same way as the other parameters using back-propagation. In practice, however, we will obtain good representations for *frequent* features with a small set of possible values, such as POS tags or characters. For word forms, it is unrealistic to find enough training data to cover every single word of a language, not to mention that all word forms have to be frequent enough in the training data for the neural model to learn good representations. Take parsing as an example. In section 4.2, we show that 37.18% of the words in our test data never appear in the training data that we use to train a dependency parser for German (table 4.1), although the training data is quite large with 720K tokens.

For that reason, it is desirable to learn good representations of words from raw text with *unsupervised* techniques that can then be used to initialize our models. That is the key idea behind word embedding algorithms today. The algorithms to train word embeddings are essentially *supervised*, but instead of relying on manual annotation, these algorithms use simple tasks for which training data can be easily created from raw text. These auxiliary tasks are also chosen with the hope that the output embeddings will possess some quality which later benefits the main tasks.

Language modeling is such an auxiliary task, and its goal is to assign a probability to a sequence of words in a language. Language models can also be used to predict the probability of a word w given its context C_w: $p(w \mid C_w)$. Formally speaking, the objective is to maximize the log-likelihood for all words in the vocabulary, given their contexts:

$$\sum_{t=1}^{|V|} \log p(w_t \mid C_{w_t}) \tag{2.28}$$

where V is the word vocabulary. Thus, neural language models can produce *similar* embeddings for words with *similar* contexts. This is in line with the *distributional hypothesis* (Harris, 1954), which states that words that occur in the same contexts tend to have similar meanings.

Neural network language models were popularized by Bengio et al. (2001). They learn an embedding matrix $\mathbf{E}$ where each column contains the representation for a word in the vocabulary:

$$\mathbf{x}_w = \mathbf{E}[w] \tag{2.29}$$

Input to the neural network is a window of size n: $w_1, w_2, ..., w_n$, and the model outputs the probability of word w_n given the context $w_1 w_2 ... w_{n-1}$. The input layer is simply the embedding concatenation of all words in the window:

$$\mathbf{x} = [\mathbf{x}_{w_1}; \mathbf{x}_{w_2}; ...; \mathbf{x}_{w_n}] \tag{2.30}$$

The model is a MLP with one hidden layer, with a softmax activation function at the output layer:

$$\mathbf{y} = \text{MLP}(\mathbf{x}) \tag{2.31}$$

or:

$$\begin{aligned} \mathbf{h} &= \tanh(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) \\ \mathbf{y} &= \text{softmax}(\mathbf{W}_2 \mathbf{h} + \mathbf{b}_2) \end{aligned} \tag{2.32}$$

where:

$$\text{softmax}(\mathbf{z})[i] = \frac{e^{\mathbf{z}[i]}}{\sum e^{\mathbf{z}[j]}} \tag{2.33}$$

The size of the output layer is $|V|$, the size of the vocabulary, and $\mathbf{y}$ corresponds to the probabilities of all words in V (including `UNK`, a symbol for unknown words). This is the computation bottleneck since the computation of softmax is proportional to $|V|$, which typically ranges from 10^5 to 10^7.

However, the neural network language model of Bengio et al. (2001) is inefficient in practice and does not produce good word representations. Based on this, Collobert and Weston (2008) propose a simpler model that eliminates the softmax layer since their ultimate goal is to learn word representations, not a language model. With the rest of the architecture similar to the previous work, their neural network model only outputs a *score* for a word given its context, and the context C_w of a word w is defined as k words to its right and k words to its left. More specifically:

$$s(w_t, C_{w_t}) = \text{MLP}(\mathbf{x}) \tag{2.34}$$

where x is the concatenation of all embeddings of w_t and C_{w_t} similar to equation 2.31. The model is trained with a margin-based ranking loss so that the score of the correct word w_t in context C_{w_t} is higher than the score of a randomly sampled word w_n given the same context:

$$\sum_{t=1}^{|V|} \sum_{w_n \in N_t} \max(0, 1 - s(w_t, C_{w_t}) + s(w_n, C_{w_t}) \tag{2.35}$$

N_t is the set of randomly sampled words, also known as *negative examples* of the word w_t. The resulting word embeddings are shown to improve the performance of many tasks (Collobert and Weston, 2008; Collobert et al., 2011). In dependency parsing, the embeddings from Collobert et al. (2011) increase the attachment scores of the first neural network parser (Chen and Manning, 2014) by 0.7%.

word2vec Pre-trained word embeddings have become the standard in NLP systems thanks to the word2vec[3] tool (Mikolov et al., 2013a; Mikolov et al., 2013b). As in previous works, the authors modify neural network language models to produce faster results. Being easy to use and efficient, word2vec helps to build pre-trained word embeddings for many languages. The tool implements two different models to learn word representations (CBOW and skip-gram), and two different training objectives (hierarchical softmax and negative sampling).

To recap, we have a neural network that computes a score for a word w_t given its context C_{w_t}: $s(w_t, C_{w_t})$. To train the language model with the log-likelihood objective (equation 2.28), we need to compute the probability for generating the word w_t:

$$p(w_t \mid C_{w_t}) = \frac{e^{s(w_t, C_{w_t})}}{\sum_{k=1}^{|V|} e^{s(w_k, C_{w_t})}} \tag{2.36}$$

Unfortunately, this formula is impractical because V is often very large. One way to approximate the softmax function is to use *hierarchical softmax*, which uses a

[3] https://code.google.com/archive/p/word2vec/

binary tree representation, thus reducing the run time from $O(|V|)$ to $O(\log_2(|V|))$. Alternatively, the log-likelihood objective can be omitted as in Collobert and Weston (2008). Mikolov et al. (2013b) propose that, instead of iterating over the whole vocabulary, uses a *negative sampling* objective where *negative examples* are randomly sampled from the data, assuming that they do, in fact, belong to the negative class. Models with the negative sampling objective are trained to *recognize* the *correct* word-context pairs (i.e., word-context pairs that originate from the corpus) and *incorrect* ones that have been randomly sampled. The objective of negative sampling is defined as:

$$\sum_{t=1}^{|V|} \left(\log P(D = 1 \mid w_t, C_{w_t}) + \sum_{w_n \in N_t} \log P(D = 0 \mid w_n, C_{w_t}) \right) \tag{2.37}$$

where N_t is the set of negative samples, and the random variable D indicates whether the word-context pair comes from the training data ($D = 1$) or not ($D = 0$). The probability that a word-context pair (w_t, C_{w_t}) is correct can be computed with the sigmoid function:

$$P(D = 1 \mid w_t, C_{w_t}) = \sigma(s(w_t, C_{w_t})) = \frac{1}{1 + e^{-s(w_t, C_{w_t})}} \tag{2.38}$$

Also:

$$P(D = 0 \mid w_t, C_{w_t}) = 1 - P(D = 1 \mid w_t, C_{w_t}) = \sigma(-s(w_t, C_{w_t})) \tag{2.39}$$

Thus, equation 2.37 becomes:

$$\sum_{t=1}^{|V|} \left(\log \sigma(s(w_t, C_{w_t})) + \sum_{w_n \in N_t} \log \sigma(-s(w_t, C_{w_t})) \right) \tag{2.40}$$

Compared to previous works, Mikolov et al. (2013a) reduce the complexity of the neural network model by removing the hidden layer and use *two* different embeddings: the *input embeddings* $\mathbf{U}$ that produce the representation $\mathbf{u}_{C_w}$ for the context C_w, and the *output embeddings* $\mathbf{V}$ that produce the representation $\mathbf{v}_w$ for the generated word w. The score of a word w_t given its context C_t is simply the dot product between two vectors:

$$s(w_t, C_{w_t}) = \mathbf{u}_{C_{w_t}}^{\top} \mathbf{v}_w \tag{2.41}$$

The *continuous bag-of-words* (CBOW) model is very similar to the language model in Collobert and Weston (2008). It also considers k words to the left and right of the word w_t as its context C_{w_t}. However, the CBOW model computes the vector of the context as the sum of all word vectors rather than using concatenation:

$$\mathbf{u}_{C_{w_t}} = \sum_{t-k \le i \le t+k, i \ne t} \mathbf{u}_{w_i} \tag{2.42}$$

The *skip-gram* model has an unconventional way to define the *context* of a word. Considering a window of size $2k + 1$ with a word w_t at the center, w_t is treated as the context to generate words $w_{t-k}, ..., w_{t-1}, w_{t+1}, ..., w_{t+k}$, resulting in $2k$ word-context pairs: $(w_{t-k}, w_t), ..., (w_{t+k}, w_t)$.

After training, the *input embeddings* of word2vec can be used as the word representations for other tasks.

Extensions of word2vec and other algorithms In word2vec, the definition of context is independent of the models and training objectives. This flexibility allows many possible modifications of the original models. The first line of work described here focuses on improving word embeddings for syntactic tasks, like POS tagging or dependency parsing. Since both the CBOW and skip-gram algorithms disregard word order, they are insensitive to syntactic properties of words such as word class, subcategorization frames, or inflection. Levy and Goldberg (2014) use the word2vec's skip-gram model to train embeddings on the English Wikipedia and do a qualitative analysis by selecting the top 5 most similar words for a target word according to the cosine similarity between their embeddings. Their examples show that the embeddings returned by word2vec have learned *topic* information: For instance, when the target word is *hogwarts* (a school of magic from the novel *Harry Potter*), the most similar words include *dumbledore*, *mafloy* and *snape* (people's names), and also *half-blood* (an adjective), all of them from the fiction series *Harry Potter*.

As an alternative, Levy and Goldberg (2014) propose to use the *dependency-based* contexts, defined as words that modify the same words (in the same dependency relation). When dependency contexts replace the original contexts in word2vec, the top similar words now belong to the same word class or have the same inflection (e.g., the *-ing* suffix) as the target word. This type of word embeddings is known as *dependency-based* word embeddings and is the top choice of word embeddings in our experiments with dependency parsing since it consistently brings small improvement over other types of embeddings. Within the same line of work, Ling et al. (2015a) modify word2vec with different embedding matrices corresponding to the position of a word in the processing window. Specifically, the authors use $2k$ input embeddings for the CBOW model (used to lookup the context embeddings), and $2k$ output embeddings for the skip-gram model (used to lookup the embeddings of the generated word), where $2k + 1$ corresponds to the processing window size in the word2vec algorithms. Their modifications are called structured word2vec (*structured CBOW* and *structured skip-gram*). Embeddings trained with the new methods improve POS tagging and dependency parsing compared to the original word2vec embeddings.

Another extension of word2vec attempts to learn *cross-lingual* embeddings that are beneficial to cross-lingual tasks. An empirical comparison of different cross-lingual embedding approaches is presented in Upadhyay et al. (2016), but here we introduce one method that directly alters the contexts of word2vec. Using a bilingual corpus, Luong et al. (2015) extend the skip-gram contexts with automatic word alignment (*bilingual skip-gram*). For instance, if there is a word-context pair (w_{t_1}, w_{c_1}) in the first language and w_{t_1} is aligned with the word w_{t_2} in the second language, the new pair (w_{t_2}, w_{c_1}) is added to the list of word-context pairs (and also the other way around).

While being trained on very large corpora, word embeddings still suffer from the *unknown word problem* because infrequent words below some predefined threshold are discarded in the training process. To overcome this, embeddings of *smaller units* are learned and combined to produce embeddings of words. *Character-based word embeddings* learn a representation for each character and use convolutional neural networks (dos Santos and Zadrozny, 2014) or RNNs (Ling et al., 2015b) to produce word embeddings. Because characters are frequent enough, character-based word embeddings can be learned together with supervised tasks. These models are very compact (because the number of characters is small) but the computation (i.e., the combination of character embeddings into word embeddings) is expensive. Character-based word embeddings produce strong results for POS tagging (dos Santos and Zadrozny, 2014; Ling et al., 2015b) and dependency parsing (Ballesteros et al., 2015).

As a compromise between characters and words, we can also compute embeddings for subwords, where word representations are decomposed into subword units, such as morpheme (Qiu et al., 2014), byte-pair encoding (Heinzerling and Strube, 2018), or n-gram (Bojanowski et al., 2017) embeddings. In chapter 4, we experiment with *compound embeddings* in syntactic parsing, where German compounds are split into their components. Other notable works on subword embeddings include fastText[4] (Bojanowski et al., 2017), a modification of the word2vec's skip-gram model to consider character n-grams. An input word w is represented as a bag of character n-grams $\mathcal{G}_w$, which contains all n-grams of w with length 3 to 6. Given a word-context pair (w_c, w_t), the original scoring function of the skip-gram model is:

$$s(w_c, w_t) = \mathbf{u}_{w_t}^\top \mathbf{v}_{w_c} \tag{2.43}$$

The fastText model replaces it with:

$$s(w_c, w_t) = \sum_{g \in \mathcal{G}_{w_t}} \mathbf{u}_g^\top \mathbf{v}_{w_c} \tag{2.44}$$

[4]https://fasttext.cc/

as now entries in the input embeddings $\mathbf{U}$ are n-grams instead of words. The embedding of a word is calculated by averaging the embeddings of its n-grams:

$$\mathbf{u}_{w_t} = \frac{1}{|\mathcal{G}_{w_t}|} \sum_{g \in \mathcal{G}_{w_t}} \mathbf{u}_g \tag{2.45}$$

Thus, fastText can provide representations of *any* word, even *out-of-vocabulary* ones.

Limitations of distributional word representations While the contribution of distributional word embeddings to neural NLP systems is undeniable, they are still far from perfect. Although word embedding algorithms group similar words together, the definition of word *similarity* is very broad. It can be topical (as in word2vec), or syntactic (as in dependency-based word embeddings). More importantly, it is the data that determine the output, while the algorithms have very little control over the kind of similarity they produce, although changing the contexts can affect the output to some extent.

Another limitation of the methods presented in this section is that they all produce one representation per word form. When updating the embeddings, the algorithms take a word form out of context, resulting in *context-independent* word representations. In practice, this is problematic since most words have multiple senses, for example, *rock* may refer to a geologic material, but can also be a genre of music. In section 2.2.4, we introduce a new family of methods to construct *context-sensitive* word representations, while in the next section, we describe how input embeddings are further processed in neural network models.

2.2.3 Feature Extraction with Bidirectional LSTMs

Given a sequence of words $w_1, w_2, ..., w_n$ where $\mathbf{x}_i$ is the input representation of the word w_i, a multilayer bidirectional LSTM (section 2.1.3) with K layers can be used to produce the hidden representation $\mathbf{h}_i$ of the word w_i. The computation of the i-th bidirectional LSTM layer (figure 2.6) is:

$$\mathbf{h}_t^{F(i)} = \text{LSTM}_F^{(i)}(\mathbf{h}_t^{F(i-1)}, \mathbf{h}_{t-1}^{F(i)}) \tag{2.46}$$

$$\mathbf{h}_t^{B(i)} = \text{LSTM}_B^{(i)}(\mathbf{h}_t^{B(i-1)}, \mathbf{h}_{t+1}^{B(i)}) \tag{2.47}$$

$$\mathbf{h}_t^{(i)} = [\mathbf{h}_t^{F(i)}; \mathbf{h}_t^{B(i)}] \tag{2.48}$$

where:

$$\mathbf{h}_t^{(0)} = \mathbf{x}_t \tag{2.49}$$

The hidden representation $\mathbf{h}_t$ is the output of the last bidirectional LSTM layer:

$$\mathbf{h}_t = \mathbf{h}_t^{(K)} \tag{2.50}$$

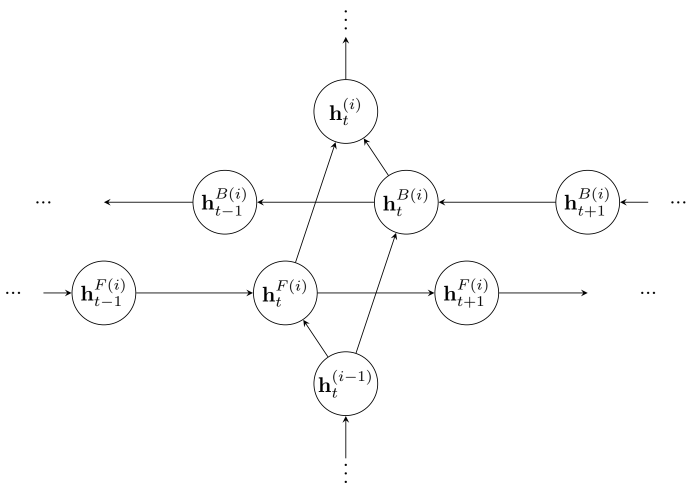

$\mathbf{h}_t^{(i)}$
$\mathbf{h}_{t-1}^{B(i)}$
$\mathbf{h}_t^{B(i)}$
$\mathbf{h}_{t+1}^{B(i)}$
$\mathbf{h}_{t-1}^{F(i)}$
$\mathbf{h}_t^{F(i)}$
$\mathbf{h}_{t+1}^{F(i)}$
$\mathbf{h}_t^{(i-1)}$

Figure 2.6: The computation at the i-th layer of a multilayer bidirectional LSTM

The input representation $\mathbf{x}_t$ can be a word embedding or the concatenation of core feature embeddings such as word and POS tag embeddings:

$$\mathbf{x}_t = [\mathbf{e}_{w_i}; \mathbf{e}_{p_i}] \tag{2.51}$$

where p_i is the POS tag of the word w_i and $\mathbf{e}$ denotes the feature embeddings.

The representations learned by bidirectional LSTMs contain information from both the left and right contexts of a word; therefore LSTMs are considered to produce word representations in context. From another perspective, bidirectional LSTMs behave like a trainable feature extraction component that extracts more abstract, high-level features from the simple input features. The hidden representations are usually used as input to higher layers in neural network models.

Multilayer bidirectional LSTMs as feature extractors have been popularized by neural sequence-to-sequence models, where they are used to *encode* information of an input sequence (Bahdanau et al., 2015). They are especially useful in sequence labeling tasks like POS tagging (Ling et al., 2015b), and are also applied in syntactic parsing (section 3.2.4).

2.2.4 Contextualized Word Embeddings

In section 2.2.2, we have introduced word embedding algorithms that represent each word form with a fixed-size vector. Although convenient, the pre-trained embeddings do not reflect well certain properties of natural languages, especially *homonymy* and *polysemy*. In section 2.2.3, we have also learned that word embeddings can be passed through several layers of bidirectional LSTMs to produce representations of words in a particular *context*. The combination of these two ideas mentioned above brings us *pre-trained* word representations that are *context-sensitive*. Implementing it is simpler than it sounds: instead of using only *embeddings* from the unsupervised task, the whole model is integrated into the supervised task of interest to provide a representation of a *word form* given its *context*.

It is no coincidence that the unsupervised task of choice to train context-sensitive (or *contextualized*) word embeddings is also language modeling. As language models are trained to predict $p(w_t \mid w_1, w_2, ..., w_{t-1})$, they need to encode the context $w_1, w_2, ...w_{t-1}$ into a fixed-size vector and use that vector to predict the next word w_t, which is considered a contextualized representation of w_t. With recurrent models (and also by using self-attention blocks in the Transformer), long-distance dependencies can be effectively captured which enables us to create high-quality contextualized word representations.

Many contextualized word embedding models have been proposed, but they are either based on LSTMs or Transformer blocks. In this section, we introduce two representative models: ELMo (Peters et al., 2018), a bidirectional LSTM language model, and BERT (Devlin et al., 2019), a multilayer bidirectional Transformer encoder. Devlin et al. (2019) classify pre-trained language representation approaches into two categories: *unsupervised feature-based approaches*, where pre-trained models are only used to provide input representations for downstream tasks, and *unsupervised fine-tuning approaches*, where the pre-trained models are integrated and their parameters are updated in a downstream task, using annotated training data. According to these definitions, ELMo is a feature-based approach[5], and BERT is a fine-tuning approach[6].

ELMo (Embeddings from Language Models) employs two language models in opposite directions, where each one is a multilayer LSTMs. Note that these language models cannot be conditioned on both directions in such a setting because a word can ßee itselfin a multilayered context. The forward language model consists of K layers of LSTMs and is trained to predict:

$$p(w_t \mid w_1, w_2, ..., w_{t-1})$$

The inputs to the forward language model are word embeddings (or character-based word embeddings) $\mathbf{x}_t$. At the k-th layer, the language model produces a context-dependent representation $\overrightarrow{\mathbf{h}}_{t,k}$ of the word w_t, and the representation at the last layer $\overrightarrow{\mathbf{h}}_{t,K}$ is used to predict w_{t+1} with a softmax layer. Similarly, the backward language

[5]Fine-tunning of ELMo means differently: the bidirectional language model is fine-tuned with data from the supervised task (with language modeling objective) without the supervision.

[6]BERT can also be used as feature-based approaches, but the common usage of BERT is with fine-tuning.

model is trained to predict:

$$p(w_t \mid w_{t+1}, w_{t+2}, ..., w_n)$$

and outputs context-dependent representations $\overleftarrow{\mathbf{h}}_{t,k}$, $k = 1...K$. The contextualized word embeddings produced by ELMo are a set of input and $2K$ hidden representations:

$$R_t = \{\mathbf{x}_t, \overrightarrow{\mathbf{h}}_{t,k}, \overleftarrow{\mathbf{h}}_{t,k} \mid k = 1...K\} \tag{2.52}$$

or:

$$R_t = \{\mathbf{h}_{t,k} \mid k = 0...K\} \tag{2.53}$$

where $\mathbf{h}_{t,0} = \mathbf{x}_t$ and $\mathbf{h}_{t,k} = [\overrightarrow{\mathbf{h}}_{t,k}; \overleftarrow{\mathbf{h}}_{t,k}]$. To use ELMo in downstream tasks, R is converted into a single vector: $\mathbf{h}_t^{\text{ELMo}} = E(R_t, \Theta^{\text{task}})$. In general, representations are combined as a weighted sum:

$$\mathbf{h}_t^{\text{ELMo}} = E(R_t, \Theta^{\text{task}}) = \gamma^{\text{task}} \sum_{k=0}^{K} s_t^{\text{task}} \mathbf{h}_{t,k} \tag{2.54}$$

where s_t^{task} are softmax-normalized weights, and γ^{task} is the scaling parameter. The parameters of ELMo are fixed and only γ^{task} and s_t^{task} are trained with the downstream model. Simply replacing traditional word embeddings with ELMo established new state-of-the-art results in 6 diverse tasks (Peters et al., 2018).

The architecture of BERT (Bidirectional Encoder Representations from Transformers) is a multilayer bidirectional Transformer (Vaswani et al., 2017) encoder (section 2.1.6). BERT is not trained on unidirectional language models as ELMo but instead uses two new unsupervised tasks. The *masked language model* objective enables language models to be conditioned in both directions, i.e., left and right contexts. In this task, some of the tokens in an input sentence are randomly masked (i.e., replaced with the token `[MASK]`) and the model is then trained to predict these tokens. In the second task, *next sentence prediction*, the model sees two input sentences, `A` and `B`, and has to predict whether `B` is likely to follow `A` in the original text. The training data for this task is generated by selecting two sentences `A` and `B` from a corpus where 50% of the time `B` succeeds `A`, and 50% of the time `B` is picked randomly from the corpus. Despite its simplicity, the next sentence prediction task is shown to benefit downstream tasks such as question answering and natural language inference (Devlin et al., 2019). To fine-tune BERT, inputs of downstream tasks are plugged directly into BERT, and the representations at the last encoder layer are used to predict the outputs with an additional classifier. All parameters are trained end-to-end. The design of BERT allows easy integration of tasks where the input is a single sentence or a pair of sentences. Fine-tuned BERT$_{\text{LARGE}}$, a model with 24 layers, 12 attention heads, and a hidden size of 1,024 (340M parameters), achieved new state-of-the-art results for 11 NLP tasks (Devlin et al., 2019). Alternatively, BERT can also be used as a feature-based approach where a downstream task cannot be transformed to BERT's architecture, or simply for computation reasons. In this case, contextualized representations are extracted from more than one hidden layer. Devlin et al. (2019) shows that the feature-based approach is only 0.3 F1 behind the fine-tuning model for name entity recognition. In section 6.4, we use BERT to replace traditional word embeddings to improve the performance of PP attachment disambiguation systems.

CHAPTER 3

Neural Dependency Parsing

In this chapter, we first introduce concepts and conventional approaches in dependency parsing (section 3.1). Then, in the second half of the chapter (section 3.2), we review how neural network techniques have transformed dependency parsing models for the last 7 years.

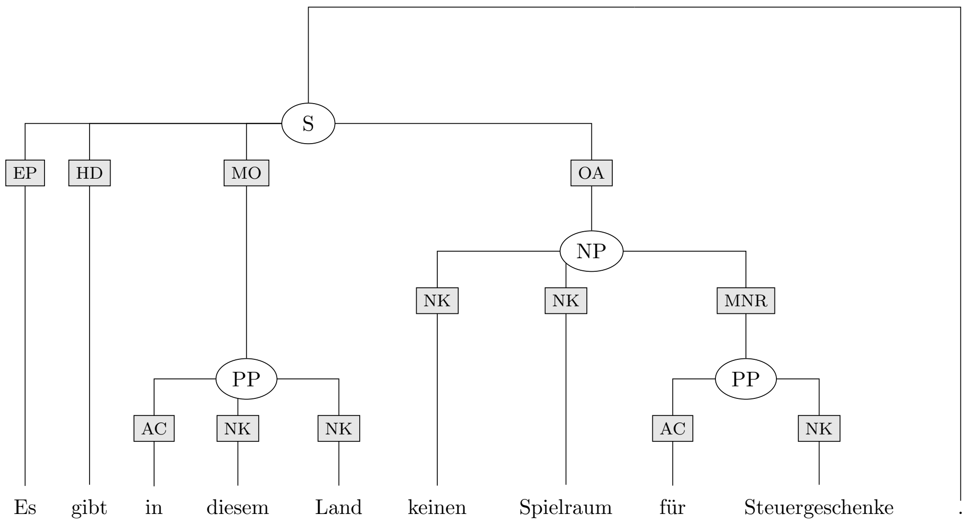
3.1 An Overview of Dependency Parsing

Syntactic parsing is the task of analyzing the *syntactic* structure of a sentence. Depending on the underlying formal assumption, the output structure can be in the form of a constituency or dependency tree (or a graph). In this section, with the main focus on *dependency parsing*, we first look at the relations between dependency parsing and its counterpart, constituency parsing (section 3.1.1). We then review the two main approaches to parse a sentence into dependency structures: transition-based (section 3.1.3) and graph-based (section 3.1.4), and compare them in section 3.1.5. Evaluation methods are introduced in section 3.1.6.

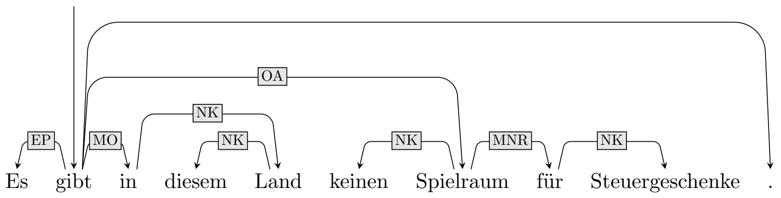
This section mainly introduces basic concepts used in the thesis. For more details on the topic, we refer the readers to Kübler et al. (2009).

3.1.1 Constituency and Dependency Parsing

Constituency parsing assigns the syntactic structure of a sentence based on *Phrase Structure Grammar* (Chomsky, 1957). Phase Structure Grammar organizes words into nested substructures called *constituents*. Figure 3.1a illustrates the constituency structure of the sentence "*Es gibt in diesem Land keinen Spielraum für Steuergeschenke.*", in which each non-terminal node, e.g. *in diesem Land*, represents a constituent. Constituenthood can be validated by *constituency tests*, but those tests are not sufficient, meaning that a true constituent may fail some of the tests. For example, only con-



(a)



(b)

Figure 3.1: The constituency structure of a German sentence *Es gibt in diesem Land keinen Spielraum für Steuergeschenke.* (There is no leeway for tax gifts in this country.) from the TIGER Treebank (a) and its corresponding (converted) dependency structure from the SPMRL 2014 Shared Task data (b). S: sentence, NP: nominal phrase, PP: prepositional phrase. AC: adpositional case marker, EP: expletive *es*, HD: head, MNR: postnominal modifier, MO: modifier, NK: noun kernel element, OA: accusative object.

stituencies can be replaced by pro-forms (e.g., it, that, do so, etc.) (*substitution*), or can be moved (such as in topicalization) (*movement*). The phrase *that tea* is a constituent in both examples below, but it fails the movement test in the second example:

(5) a. I like <u>that tea</u>.
  b. Substitution: I like <u>it</u>.
  c. Movement: <u>That tea</u>, I like __.

(6) a. I like cakes and <u>that tea</u>.
  b. Substitution: I like cakes and <u>it</u>.
  c. * Movement: <u>That tea</u>, I like cakes and __.

Dependency parsing, on the other hand, is based on *Dependency Grammar* (Mel'čuk, 1988; Tesnière, 1959). Lying at the heart of Dependency Grammar is the binary, asymmetric relations between words in a sentence, called *dependency relations* (or *dependencies*). A dependency relation represents a syntactic relation between a word, called the *dependent*, and another word that it depends on, called the *head*. In figure 3.1b, the dependency structure of the previous German sentence is represented by directed arcs pointing from heads to dependents. The label on each arc indicates the *dependency type*, or the *grammatical function* of the relation between two words. For instance, the dependency label of the arc $gibt \to Spielraum$ is *OA*, which means *accusative object*. The head of the word *gibt* in the figure is undefined, or more precisely, its head is not any word in the sentence. Technically, the word *gibt* is the root of the dependency tree, but in dependency parsing, we consider that its head is the artificial root ROOT. Later, we will see that this root token is inserted at the beginning of the sentence, and is considered a part of the sentence in formal definitions.

Although based on different grammar formalisms, the information encoded in constituency and dependency structures are not mutually exclusive. For example, a subtree in a dependency tree has a similar function as to a constituent, whereas a constituent can be converted into dependencies by first determining a head, and then attaching other components that depend on it. In fact, many dependency treebanks are converted from existing constituent treebanks with head-finding rules (Magerman, 1994) to determine the head of a constituent. The task to convert constituent treebanks into dependency ones is not straightforward since linguists tend to disagree on what should be considered as heads (content vs. function words), especially for cases like auxiliary verbs, prepositional phrases, coordination, etc.

In recent years, dependency parsing has surpassed constituency parsing and has attracted considerable attention in natural language processing (NLP). One reason for that is the simplicity of dependency over constituency structures, making it possible to analyze languages with free or flexible word order with minimal changes. Because of that, Universal Dependencies (UD), a common framework for treebank development in dependency format, has been a great success, as the UD framework also helps to facilitate cross-lingual research in dependency parsing. In addition, the head-dependent encoding is closer to the predicate-argument semantic relations than the phrase structures, thus the representations are particularly useful for downstream applications such as question answering, information extraction, machine translation, etc. Lastly, the development of fast and accurate dependency parsers with the help of neural network methods might be an important reason for the popularity of dependency parsing.

3.1.2 Definitions

Definition 3.1. In dependency parsing, a *sentence* is a sequence of tokens:

$$S = w_0 w_1 ... w_n$$

where $w_0 = \text{ROOT}$ is an artificial root token appended at the beginning of the sentence.

In the most general form, the output dependency structure for a sentence is a directed graph.

Definition 3.2. Given a set of dependency labels R, a *dependency graph* for a sentence $S = w_0 w_1 ... w_n$ is a directed graph $G = (V, A)$, where:

1. $V \subseteq V_S = \{w_0, w_1, ..., w_n\}$ is the set of nodes,
2. $A \subseteq V \times R \times V$ is the set of dependency arcs,
3. there is only one arc between two nodes, or if $(w_i, r, w_j) \in A$ then s$(w_i, r', w_j) \notin A$, $\forall r' \in R$.

We denote $V_S = \{w_0, w_1, ..., w_n\}$ as the set of nodes containing all tokens in the sentence S.

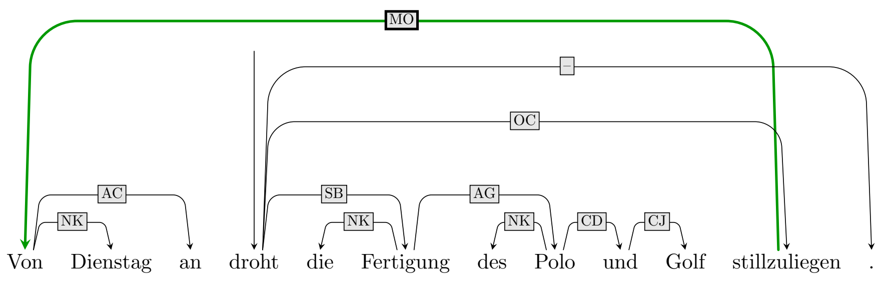
For many languages, however, we want the output to be a tree.

Definition 3.3. A dependency graph $G = (V, A)$ is a *dependency tree* if it satisfies the following conditions:

1. w_0 is the root that has no incoming arcs: $\nexists i, r$ such as $(w_i, r, w_0) \in A$.
2. Every other node has exactly one incoming arc: $\forall j \exists i, r : (w_i, r, w_j) \in A$, and $\nexists i' \neq i, r' : (w_{i'}, r', w_j) \in A$
3. There is a unique path from the root w_0 to each node in V. This also means that G is acyclic and connected.

Notation 3.1. The notation $w_i \to w_j$ indicates the *unlabeled dependency relation* in a tree $G = (V, A)$. That is, $w_i \to w_j$ if and only if $(w_i, r, w_j) \in A$ for some r.

Notation 3.2. The notation $w_i \to^* w_j$ indicates the *reflexive transitive closure* of the dependency relation in a tree $G = (V, A)$. That is, $w_i \to^* w_j$ if and only if $i = j$ (reflexive) or $\exists i'$ such as $w_i \to^* w_i'$ and $w_i' \to w_j$ (transitive).


MO
–
OC
AC
NK
SB
NK
AG
NK
CD
CJ
Von Dienstag an droht die Fertigung des Polo und Golf stillzuliegen .

"From Tuesday onward, the production of the Polo and Golf threatens to be shut down."

Figure 3.2: A non-projective dependency tree from the SPMRL 2014 Shared Task data. The non-projective arc is marked with a thick line.

$w_i \rightarrow^* w_j$ means that w_j is reachable from w_i.

An interesting phenomenon in languages is *projectivity*. Intuitively, when drawing a dependency tree of a sentence with regards to the word order, the tree is *projective* if each arc does not intersect other arcs (except at end points). If there are crossing arcs, the tree is *non-projective*. The dependency tree in figure 3.1b is projective, while the one in figure 3.2 is non-projective. Non-projectivity is often encountered in languages with free or flexible word order and long dependencies like Czech or German, and less so in languages with configurational word order like English and Chinese.

An arc from a head to a dependent is projective if all nodes between them are reachable from the head.

Definition 3.4. An arc $(w_i, r, w_j) \in A$ in a dependency tree $G = (V, A)$ is *projective* if and only if $w_i \rightarrow^* w_k$ for all $i < k < j$ when $i < j$, or $j < k < i$ when $j < i$.

A dependency tree is projective if all of its arcs are projective. Otherwise, it is non-projective.

Definition 3.5. A dependency tree $G = (V, A)$ is *projective* if and only if for all $(w_i, r, w_j) \in A$ is projective.

Given training data in the form of a treebank, the goal of dependency parsing is to predict the correct dependency tree for an input sentence S. There exist two main ways to model the dependency parsing problem. The first approach learns the best sequence of actions to build a dependency tree from scratch and is called *transition-based* parsing. The second approach tries to find the best spanning tree over the complete graph of all the words in the sentence using well-known graph and tree algorithms, thus is known as *graph-based* parsing. The next sections will give an overview of these two approaches.

3.1.3 Transition-Based Parsing

The procedure to construct a dependency tree in transition-based parsing is defined by a *transition system*. Like an abstract machine, a transition system consists of a set of states, or *configurations*, and can be changed from one configuration to another using a *transition*. In the scope of this chapter, we mainly focus on stack-based transition systems, in which a configuration is defined by a stack, a buffer, and a set of dependency arcs.

Definition 3.6. Given a set of dependency types R, a *configuration* for a sentence $S = w_0w_1...w_n$ is a triple $c = (\sigma, \beta, A)$ where:

1. σ is a stack of words $w_i \in V_S$,
2. β is a buffer of words $w_i \in V_S$,
3. A is a set of dependency arcs $(w_i, r, w_j) \in V_S \times R \times V_S$.

The stack represents partially processed nodes that can potentially be linked to other nodes as heads or dependents. The buffer contains the remaining unprocessed nodes, and the arc set is actually a partially constructed dependency tree. Both the stack and the buffer are represented as lists, where the top of the stack is on the right and the head of the buffer is on the left of the list. Thus, $[\sigma|w_i]_\sigma$ denotes a stack with token w_i on the top, and $[w_j|\beta]_\beta$ denotes a buffer with token w_j as the head.

In the beginning, the root token is pushed onto the stack, and the rest of the tokens are in the buffer. The transition system terminates when all tokens are processed, i.e., when the buffer is empty.

Definition 3.7. For a sentence $S = w_0w_1...w_n$:

1. The *initial configuration* c_0 is $([w_0]_\sigma, [w_1, ..., w_n]_\beta, \emptyset)$,
2. A *terminal configuration* $c_t \in C_t$ has the form $(\sigma, [\,]_\beta, A)$ for any σ and A.

Formally speaking, a *transition* is a (partial) function defined on the set of all possible configurations $\mathcal{C}$: $t : \mathcal{C} \rightarrow \mathcal{C}$. In stack-based parsing, a transition can manipulate the stack and the buffer (to pop or push words from/to the stack, to add or remove words from/to the buffer) and add a new arc to the set of arcs. Different transitions utilize those actions in different ways and combining different types of transitions results in different parsing strategies, or transition systems.

Definition 3.8. A *transition system* is a quadruple $\mathcal{S} = (\mathcal{C}, \mathcal{T}, c_I, C_t)$ where:

1. $\mathcal{C}$ is a set of configurations defined in definition 3.6,

Transitions		Preconditions
LEFT-ARC$_r$	$(\sigma\|w_i, w_j\|\beta, A) \Rightarrow (\sigma, w_j\|\beta, A \cup \{(w_j, r, w_i)\})$	$i \neq 0$
RIGHT-ARC$_r$	$(\sigma\|w_i, w_j\|\beta, A) \Rightarrow (\sigma, w_i\|\beta, A \cup \{(w_i, r, w_j)\})$	
SHIFT	$(\sigma, w_i\|\beta, A) \Rightarrow (\sigma\|w_i, \beta, A)$	

Figure 3.3: Transitions for the arc-standard algorithm

2. $\mathcal{T}$ is a set of transitions, each is a partial function $t : \mathcal{C} \rightarrow \mathcal{C}$,

3. c_I is an initialization function that maps sentence S to its initial configuration: $c_0 = c_I(S)$,

4. $C_t \subseteq \mathcal{C}$ is a set of terminal configurations.

One of the popular transition systems, arc-standard (Nivre, 2004; Nivre, 2008), is defined in figure 3.3. In this transition system, the LEFT-ARC$_r$ transition for any relation r creates a left arc (w_j, r, w_i) between the node at the top of the stack w_i and the node at the beginning of the buffer w_j. After that, the element at the top of the stack is popped out. The only precondition to these actions is w_i is not the root. Similarly, the RIGHT-ARC$_r$ transition for any relation r creates a right arc (w_i, r, w_j) between the node at the top of the stack w_i and the node at the beginning of the buffer w_j. After that, w_i is popped from the stack and replaces w_j at the beginning of the buffer. The SHIFT transition removes the first word in the buffer and puts it on the top of the stack. It is important to note that the arc-standard system only works for *projective* trees.

Following Bohnet and Nivre (2012), we define a transition sequence as a sequence of configuration-transition pairs:

Definition 3.9. A *transition sequence* for a sentence $S = w_0w_1...w_n$ is a sequence of configuration-transition pairs $C_{0,m} = [(c_0, t_0), (c_1, t_1), ..., (c_m, t_m)]$ such that:

1. c_0 is the initial configuration $c_I(S)$ for S,

2. $t_m(c_m) \in C_t$ is a terminal configuration,

3. for every $0 \leq i < m$, $t_i \in \mathcal{T}$ and $t_i(c_i) = c_{i+1}$.

Figure 3.4 illustrates an arc-standard transition sequence to parse the tree in figure 3.1b.

Transition	Configuration		
	σ	β	A
	[ROOT]	[Es, ...]	$\emptyset$
SH	[ROOT, Es]	[gibt, ...]	$\emptyset$
LA_{EP}	[ROOT]	[gibt, ...]	$A_1 = \{(\text{gibt, EP, Es})\}$
SH	[ROOT, gibt]	[in, ...]	A_1
SH	[ROOT, gibt, in]	[diesem, ...]	A_1
SH	[ROOT, ..., diesem]	[Land, ...]	A_1
LA_{NK}	[ROOT, gibt, in]	[Land, ...]	$A_2 = A_1 \cup \{(\text{Land, NK, diesem})\}$
RA_{NK}	[ROOT, gibt]	[in, ...]	$A_3 = A_2 \cup \{(\text{in, NK, Land})\}$
RA_{MO}	[ROOT]	[gibt, ...]	$A_4 = A_3 \cup \{(\text{gibt, MO, in})\}$
SH	[ROOT, gibt]	[keinen, ...]	A_4
SH	[ROOT, gibt, keinen]	[Spielraum, ...]	A_4
LA_{NK}	[ROOT, gibt]	[Spielraum, ...]	$A_5 = A_4 \cup \{(\text{Spielraum, NK, keinen})\}$
SH	[ROOT, gibt, Spielraum]	[für, ...]	A_5
SH	[ROOT, ..., für]	[Steuergeschenke, .]	A_5
RA_{NK}	[ROOT, gibt, Spielraum]	[für, .]	$A_6 = A_5 \cup \{(\text{für, NK, Steuergeschenke})\}$
RA_{MNR}	[ROOT, gibt]	[Spielraum, .]	$A_7 = A_6 \cup \{(\text{Spielraum, MNR, für})\}$
RA_{OA}	[ROOT]	[gibt, .]	$A_8 = A_7 \cup \{(\text{gibt, OA, Spielraum})\}$
SH	[ROOT, gibt]	[.]	A_8
$\text{RA}_{\text{-}}$	[ROOT]	[gibt]	$A_9 = A_8 \cup \{(\text{gibt, –, .})\}$
$\text{RA}_{\text{-}}$	[ ]	[ROOT]	$A_{10} = A_9 \cup \{(\text{ROOT, –, gibt})\}$
SH	[ROOT]	[ ]	A_{10}

Figure 3.4: Arc-standard transition sequence for the sentence in figure 3.1b. LA_r, RA_r, and SH stand for LEFT-ARC$_r$, RIGHT-ARC$_r$, and SHIFT.

Algorithm 1: Greedy transition-based parsing

```
1 function PARSE_greedy($(x)$)
2     $c \leftarrow c_I(x)$
3     while $c \notin C_t$ do
4         $t_p \leftarrow \arg\max_{t \in \text{VALID}(c)} score(c, t)$
5         $c \leftarrow t_p(c)$
6     end
7     return $A_c$
```

Decoding Given a sentence S, a transition-based parsing system should return the most probable sequence of transitions to build a dependency tree, indicated by a scoring function:

$$\arg\max_{C_{0,m}} score(S, C_{0,m}) \tag{3.1}$$

Algorithm 2: Beam search for transition-based parsing

```
function PARSE_beam($(x)$)
    $h_0.c \leftarrow c_I(x)$
    $h_0.s \leftarrow 0$
    BEAM $\leftarrow [h_0]$
    while $\exists h \in$ BEAM : $h.c \notin C_t$ do
        TMP $\leftarrow [\,]$
        foreach $h \in$ BEAM do
            foreach $t \in$ VALID($h.c$) do
                $h'.s \leftarrow h.s + score(h.c, t)$
                $h'.c \leftarrow t(h.s)$
                PUSH(TMP, $h'$)
            end
        end
        BEAM $\leftarrow$ PRUNE(TMP)
    end
    $h \leftarrow$ TOP(BEAM)
    return $A_{h.c}$
```

The score of a transition sequence can be decomposed as the sum of individual transition scores:

$$score(S, C_{0,m}) = \sum_{(c,t) \in C_{0,m}} score(c, t) \tag{3.2}$$

Assuming that we already have such a scoring function, transition-based parsing can be performed as a simple *greedy* strategy, where in each step the transition with the highest score is followed (algorithm 1). Here, VALID(c) returns a set of permissible transitions at a configuration c according to the transition system. With a transition system like arc-standard, the worst time and space complexity is a linear function of n, the length of the input sentence (Nivre, 2008). If scoring can be done in constant time, the time complexity of the greedy parsing algorithm will also be $O(n)$. Although the greedy algorithm is easy to implement and efficient, it is prone to *error propagation*. Once an incorrect decision is made, more errors are likely to follow if the parser has no mechanism to recover from previous errors. Error propagation will be discussed more later when we discuss how to train a transition-based parser (or, how to learn the scoring function), but one way to reduce it is to incorporate a *beam* (algorithm 2). Because we always keep a fixed beam size (function PRUNE in algorithm 2), the time complexity of the beam search decoding algorithm is still linear. While the greedy

$$o(c) = \begin{cases} \textsc{Left-Arc}_r & c = (\sigma|w_i, w_j|\beta, A) \text{ and } (w_j, r, w_i) \in A_d \\ \textsc{Right-Arc}_r & c = (\sigma|w_i, w_j|\beta, A), (w_i, r, w_j) \in A_d \\ & \text{and for all } w, r' \text{ if } (w_j, r', w) \in A_d \text{ then } (w_j, r', w) \in A \\ \textsc{Shift} & \text{otherwise} \end{cases}$$

Figure 3.5: Oracle function for the arc-standard transition system for the target tree $G_d = (V_d, A_d)$.

and beam search algorithms are instances of inexact search, exact decoding using dynamic programming has also been implemented for transition-based parsing by decomposing a sequence of computations into smaller, shareable parts (Kuhlmann et al., 2011).

Oracle To learn the scoring function, we first need a guide to tell what is the best transition to take at the current configuration. Such a guide is called an *oracle* and is a function that maps configurations to transitions: $o : \mathcal{C} \to \mathcal{T}$. We say a tree G is *reachable* from a configuration c if after applying a finite number of transitions, the tree built in c is G. Given a sentence S and its correct parse tree G_d, an oracle function can be designed to return one correct transition t for any configuration c so that G_d can be reached from $t(c)$. The oracle function for the arc-standard system is shown in figure 3.5. The oracle returns the correct transition as LEFT-ARC$_r$ if there is an arc from the token at the beginning of the buffer w_j to the token w_i on top of the stack in the correct tree. The correct transition is RIGHT-ARC$_r$ if there is an arc with head w_i and dependent w_j, but with an extra condition that all outgoing arcs from w_j have already been constructed. This condition is necessary because once a RIGHT-ARC transition is made, w_j will be removed from the buffer and can no longer take any dependents. However, this is not required in the case of a LEFT-ARC transition because it is automatically satisfied if the target tree is *projective*.

Being trained using an oracle, the scoring function is actually an approximation of the oracle's behavior and is one way to formulate the problem. An alternative way is to learn the oracle's decisions (i.e., the correct transition) directly with a discriminative classifier.

Features In order to use machine learning techniques to learn a parsing oracle, the input needs to be mapped to a feature representation space. Here, we want to learn a function $\mathbf{f}(c)$ that maps an input configuration c to a k-dimensional one-hot vector:

Token	Attribute				
	Word	POS tag	Valency	Dependency label	Label set
First order					
STK[0]	$*$	$*$	+	+	+
BUF[0]	$*$	$*$	+		+
BUF[1]	$*$	$*$			
BUF[2]	$*$	$*$			
Second order					
HEAD(STK[0])	+	$*$		+	
LDEP(STK[0])	+	$*$		+	
RDEP(STK[0])	+	$*$		+	
LDEP(BUF[0])	+	$*$		+	
Third order					
HEAD(HEAD(STK[0]))	+	+			
LDEP_2(STK[0])	+	+		+	
RDEP_2(STK[0])	+	+		+	
LDEP_2(BUF[0])	+	+		+	

Table 3.1: *Core* features for transition-based parsing used in Zhang and Nivre (2011): $*$: baseline feature, +: extended feature. The distance (between STK[0] and BUF[0]) feature and feature combinations are not listed.

$\mathbf{f} : \mathcal{C} \rightarrow \{0, 1\}^k$. We use STK[$i$] and BUF[$i$] to refer to the i-th word from the top of the stack and the beginning of the buffer respectively. HEAD(w), LDEP(w), and RDEP(w) denote the head, the leftmost modifier, and the rightmost modifier of a word w. In a similar fashion, $\text{LDEP}_2(w)$ and $\text{RDEP}_2(w)$ refer to the second leftmost and rightmost modifiers of a word w. Baseline features for traditional transition-based parsers (Zhang and Clark, 2008; Zhang and Nivre, 2011) include *words* and *POS tags* from:

- the word at the top of the stack STK[0],
- the first three words in the buffer BUF[0], BUF[1], and BUF[2]
- the head, the leftmost and rightmost modifier of STK[0],
- the leftmost modifier of BUF[0].

Zhang and Nivre (2011) show that the parsing accuracy can be improved by considering a richer and non-local feature set which includes additional attributes (*valency*,

Algorithm 3: Local online training for transition-based parsers

```
input  : set of training examples $\{(x_i, y_i)\}_{i=1}^N$
         $x_i$ is a sentence and $y_i$ is the dependency tree of $x_i$
output : weight matrix $\mathbf{W}$
1  $\mathbf{W} \leftarrow 0$
2  for $k \in 1..K$ do
3      foreach training example $(x_i, y_i)$ do
4          $c \leftarrow c_I(x_i)$
5          while $c \notin C_t$ do
6              $t_p \leftarrow \arg\max_{t \in \text{VALID}(c)} \mathbf{f}(c) \cdot \mathbf{W}^t$
7              $t_o \leftarrow o(c, y_i)$
8              if $t_p \neq t_o$ then
9                  UPDATE($\mathbf{W}^{t_o}, \mathbf{W}^{t_p}, \mathbf{f}(c)$)
10             end
11             $c \leftarrow t_o(c)$
12         end
13     end
14 end

1  function UPDATE($\mathbf{W}^{t_o}, \mathbf{W}^{t_p}, \mathbf{f}(c)$)
2      $\mathbf{W}^{t_o} \leftarrow \mathbf{W}^{t_o} + \mathbf{f}(c)$
3      $\mathbf{W}^{t_p} \leftarrow \mathbf{W}^{t_p} - \mathbf{f}(c)$
```

dependency label, *label set*) and third-order relations (the grandparent of STK[0], the second leftmost modifier of STK[0] and BUF[0], the second rightmost modifier of STK[0]). Table 3.1 summarizes the baseline and extended feature sets for transition-based dependency parsing.

Training After defining a feature function, a variety of machine learning techniques can be applied to predict the correct transition for a configuration. Two approaches that have been widely used for transition-based dependency parsing are memory-based learning (Nivre, 2003) and support vector machines (Yamada and Matsumoto, 2003; Nivre et al., 2006b). Alternatively, the score of a transition can be computed as the dot product of the feature vector $\mathbf{f}(c)$ and a transition-specific weight vector $\mathbf{w}_t$:

$$score(c, t) = \mathbf{f}(c) \cdot \mathbf{w}_t \tag{3.3}$$

We can stack all transition-specific weight vectors into a weight matrix:

$$\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_{|\mathcal{T}|}] \tag{3.4}$$

Algorithm 4: Structured perceptron training algorithm (global training) for transition-based parsers

```
input  : set of training examples $\{(x_i, y_i)\}_{i=1}^{N}$
         $x_i$ is a sentence and $y_i$ is the dependency tree of $x_i$
output : weight matrix $\mathbf{W}$
1 $\mathbf{W} \leftarrow 0$
2 for $k \in 1..K$ do
3     foreach training example $(x_i, y_i)$ do
4         $y_p \leftarrow \arg\max_{y \in \text{GEN}(x)} \sum_{(c,t) \in \textsc{TranSeq}(y)} \mathbf{f}(c) \cdot \mathbf{W}^t$
5         if $y_p \neq y$ then
6             $\textsc{Update}(\mathbf{W}, y, y_p)$
7         end
8     end
9 end

1 function $\textsc{Update}(\mathbf{W}, y, y_p)$
2     foreach $(c, t) \in \textsc{OracleTranSeq}(y)$ do
3         $\mathbf{W}^t \leftarrow \mathbf{W}^t + \mathbf{f}(c)$
4     end
5     foreach $(c, t) \in \textsc{TranSeq}(y_p)$ do
6         $\mathbf{W}^t \leftarrow \mathbf{W}^t - \mathbf{f}(c)$
7     end
```

and $\mathbf{W}^t$ denotes the t-th column of $\mathbf{W}$, also corresponding to $\mathbf{w}_t$. A basic online algorithm to learn $\mathbf{W}$ is shown in algorithm 3. Similar to support vector machines, this approach is *local* because each transition decision is optimized independently. Training a parser locally makes error propagation more severe since the parser never sees bad configurations in the training process. Interestingly, Zhang and Nivre (2012) empirically show that local parsers do not benefit from beam search and non-local features. In fact, beam search hurts the parser's performance in comparison to greedy decoding because of unseen negative examples in the beam. This reason can be generalized to the mismatch between training and testing conditions, and the authors also show that the accuracy of (globally trained) parsers is better if the beam sizes at training and testing time are the same. From another perspective, beam search has a negative effect on local parsing models because those models are trained to disambiguate different transitions under the same configuration, but not different transitions under different parser configurations.

The perceptron algorithm (Collins, 2002) is a widely used method for structured prediction problems and has also been applied to transition-based parsing (Zhang and Clark, 2008; Zhang and Nivre, 2011). Algorithm 4 illustrates a version of the perceptron algorithm for transition-based parsing. Instead of updating the weight vectors for individual transitions, the algorithm optimizes them over the entire sequence of transitions, thus the system is trained with *global* information. A component of the algorithm is GEN(x), a function that generates candidate predictions, and y_p is the tree with the highest score among them. In practice, y_p can be computed using exact decoding based on dynamic programming, or approximated with beam search. TRANSEQ(y) returns the transition sequence of y via decoding, and in a similar manner, ORACLETRANSEQ(y) returns the transition sequence guided by the oracle. The original algorithm from Collins and Koo (2005) uses exact search because it is guaranteed to converge. For a normal full update using beam search, the gold transition sequence may fall out of the beam, but it is still scored higher than any item in the beam. This would lead to invalid updates and exact search is one way to ensure this does not happen. Collins and Roark (2004) use *early update*, i.e., they update the parameters as soon as the gold sequence falls out of the beam. Huang et al. (2012) present a general framework for *violation-fixing* perceptron and propose several new update mechanisms including the *max-violation update*. Unlike local models, beam search and non-local features have a positive impact on the global model (Zhang and Nivre, 2012).

Non-projective parsing Many stack-based transition systems, including the arc-standard algorithm that served as an example in this section, restrict the input tree to be projective. Nevertheless, many languages and linguistic constructions require a non-projective analysis. However, while the amount of non-projective trees in some dependency treebanks is about 10-25%, the proportion of non-projective arcs is only 1-2% (Nivre and Nilsson, 2005), which means that evaluation metrics that average scores over all edges in the tree do not give enough attention to non-projective tree structures. The attractive linear running complexity will also suffer when moving from projective to non-projective transition-based parsing since the number of arcs to consider is $n(n-1) = O(n^2)$ without any restriction. One of the early works along those lines is *pseudo-projective parsing* (Nivre and Nilsson, 2005), which combines projective transition-based parsing and graph transformation to parse non-projective structures. The approach performs four steps:

1. Projectivize a dependency graph (i.e., convert a non-projective dependency graph into a projective one by lifting non-projective arcs) while retaining the original structure as much as possible and encoding the transformation information in the augmented arc labels

Transitions		Preconditions
LEFT-ARC$_r$	$(\sigma\|w_i, w_j\|\beta, A) \Rightarrow (\sigma, w_j\|\beta, A \cup \{(w_j, r, w_i)\})$	$i \neq 0$
RIGHT-ARC$_r$	$(\sigma\|w_i, w_j\|\beta, A) \Rightarrow (\sigma, w_i\|\beta, A \cup \{(w_i, r, w_j)\})$	
SHIFT	$(\sigma, w_i\|\beta, A) \Rightarrow (\sigma\|w_i, \beta, A)$	
SWAP	$(\sigma\|w_i, w_j\|\beta, A) \Rightarrow (\sigma, w_j w_i\|\beta, A)$	0 < i < j

Figure 3.6: Transitions for the swap-standard algorithm

2. Train a projective parser on the projectivized training data

3. Parse the projectivized test data with the parser

4. Deprojectivize the newly parsed sentences with guides from dependency arc labels

Pseudo-projective parsing can be combined with any projective parsing algorithm, but it is sensitive to the amount of training data (because the transformation introduces many new labels) and the complexity of non-projective constructions (number of lifts to convert non-projective arcs to projective ones) (Nilsson et al., 2007).

We have learned from section 3.1.2 that the definition of projectivity is connected to the word order in a sentence. Thus, any tree can be made projective by reordering the words. Based on this idea, Nivre (2009) proposes to swap words simultaneously while constructing dependency arcs. The author augments the arc-standard system with a new transition SWAP that swaps the two adjacent words at the top of the stack and the head of the buffer (figure 3.6). The new transition system, sometimes referred to as swap-standard, defines an oracle function to return the SWAP action when the *projective order* of the two nodes (at the stack top and the beginning of the buffer) is not satisfied. The projective order of a dependency tree G is the order of its nodes when reordering them to make G projective. Later, Nivre et al. (2009) design a new oracle for swap-standard that delays the SWAP action until necessary using the concept of *maximal projective components*.

Non-deterministic oracle From one configuration, there is usually more than one transition to be taken, but instead, the oracle that trains a parser only chooses one of them. It also entails that there is only one *unique* optimal transition sequence to build any tree. Not only it makes parsing inflexible, but also a parser trained with this type of oracles is more likely to suffer from error propagation when the deduced transition sequence starts to diverse from the gold standard one. Non-local features, global training, and beam search, all induce a global view to a parser, but still, the

Algorithm 5: Local online training using a non-deterministic oracle for transition-based parsers

```
input  : set of training examples $\{(x_i, y_i)\}_{i=1}^N$
         $x_i$ is a sentence and $y_i$ is the dependency tree of $x_i$
output : weight matrix $\mathbf{W}$
1  $\mathbf{W} \leftarrow 0$
2  for $k \in 1..K$ do
3      foreach training example $(x_i, y_i)$ do
4          $c \leftarrow c_I(x_i)$
5          while $c \notin C_t$ do
6              $t_p \leftarrow \arg\max_{t \in \text{VALID}(c)} \mathbf{f}(c) \cdot \mathbf{W}^t$
7              $T_o \leftarrow o_N(c, y_i)$
8              if $t_p \notin T_o$ then
9                  UPDATE$(\mathbf{W}^{t_o}, \mathbf{W}^{t_p}, \mathbf{f}(c))$
10                 $t_n \leftarrow t_o;\ t_o \in T_o$
11             else
12                 $t_n \leftarrow t_p$
13             end
14             $c \leftarrow t_n(c)$
15         end
16     end
17 end
```

parser is optimized on one single transition sequence per sentence. Given a sentence S and its corresponding dependency tree G, suppose that we want to train a parser on all possible transition sequences that lead to G. For that, we need a new type of oracle that returns a set of transitions rather than a single transition. We call this type of oracle *non-deterministic* (Goldberg and Nivre, 2013), and in contrast, the old type *static oracle*. Given a configuration c, a non-deterministic oracle function o_N should return all transitions t so that the tree G is reachable from $t(c)$. We informally refer to a transition that leads to the gold standard tree as a *correct* transition. The online algorithm to train a parser with a non-deterministic oracle (algorithm 5) is different from the previous one with a static oracle (algorithm 3) in one point: it is non-deterministic because when the predicted transition t_p is incorrect, the algorithm *randomly* selects a correct transition to follow. Björkelund and Nivre (2015) design two versions of non-deterministic oracles for the swap-standard system, and show that non-deterministic oracles consistently improve a greedy, local parser on 10 languages,

Algorithm 6: Local online training with exploration using a dynamic oracle for transition-based parsers

```
input  : set of training examples $\{(x_i, y_i)\}_{i=1}^N$
         $x_i$ is a sentence and $y_i$ is the dependency tree of $x_i$
output : weight matrix $\mathbf{W}$
1  $\mathbf{W} \leftarrow 0$
2  for $k \in 1..K$ do
3      foreach training example $(x_i, y_i)$ do
4          $c \leftarrow c_I(x_i)$
5          while $c \notin C_t$ do
6              $t_p \leftarrow \arg\max_{t \in \textsc{Valid}(c)} \mathbf{f}(c) \cdot \mathbf{W}^t$
7              $T_o \leftarrow o_D(c, y_i)$
8              if $t_p \notin T_o$ then
9                  $\textsc{Update}(\mathbf{W}^{t_o}, \mathbf{W}^{t_p}, \mathbf{f}(c))$
10                 $t_n \leftarrow \textsc{Explore}(\mathcal{T}, T_o, t_p)$
11             else
12                 $t_n \leftarrow t_p$
13             end
14             $c \leftarrow t_n(c)$
15         end
16     end
17 end
```

but the effect is mixed when applied to a global parser with beam search.

What if an oracle can tell us an optimal transition even when the gold tree is not reachable? This is the idea behind the *dynamic* oracle, or an oracle that is both non-deterministic and *complete* (in the sense that it still functions when the correct tree can no longer be reached). The formal definition and more details about non-deterministic and dynamic oracles can be found in Goldberg and Nivre (2013). Suppose we have a dynamic oracle, the training algorithm for it (algorithm 6) is very similar to the one for a non-deterministic oracle, except that the next transition to follow, t_n, is no longer required to be a correct one. This step is the exploration, denoted by function EXPLORE that considers the set of correct transitions T_o, the predicted transition t_p, and the set of all transitions $\mathcal{T}$. There exists different exploration schemes, for example, Goldberg and Nivre (2013) alternate between the highest score transition in T_o and t_p. They also postpone following an incorrect decision until after some iterations, but state that the parameters are insensitive and the training scheme works as long as the predicted transition is chosen often. However, with a more powerful classifier such as neural networks that converge much faster, Ballesteros et al. (2016) observe that the predicted transition t_p quickly becomes accurate and the parser is rarely exposed to incorrect configurations during training. Rather, they propose to sample the next transition from the predicted distribution of the classifier.

The remaining question is how to construct a dynamic oracle. Basically, a dynamic oracle $o_N(c, G)$ is defined to return all transitions t so that the best tree reachable from $t(c)$ is not worse than the best tree reachable from c. We denote $c \rightsquigarrow G$ as tree G is reachable from c. The statement above can be defined in terms of the *cost* difference $\mathcal{C}(G, G')$ between a tree G' and the gold tree G. Specifically:

$$o_N(c, G) = \{t \mid \min_{G':t(c)\rightsquigarrow G'} \mathcal{C}(G, G') - \min_{G':c\rightsquigarrow G'} \mathcal{C}(G, G') = 0\} \tag{3.5}$$

The transitions above can be called *zero cost* transitions, and they can be efficiently determined if the transition system is *arc decomposable* (Goldberg and Nivre, 2012). Unfortunately, arc decomposition does not hold for the arc-standard system, and the dynamic oracle for it can be derived using dynamic programming (Goldberg et al., 2014) with worst-case complexity $O(n^3)$, although in practice, the dynamic programming algorithm runs only 2.3 times slower than the static oracle. However, a dynamic oracle for other transition systems including arc-eager (Nivre, 2003) and arc-hybrid (Kuhlmann et al., 2011) can be derived using arc decomposition.

3.1.4 Graph-Based Parsing

Dependency parsing can be naturally formulated as a graph problem. Given a sentence S and its node set V_S, a dependency tree G is actually a *spanning tree* of the full (with an exception for the root node, which only has outgoing arcs), directed graph on V_S. If we can give each tree a score to indicate how likely it is that the tree represents the correct analysis of S, then dependency parsing becomes finding the highest scored tree in the set of all possible trees $\mathcal{G}_S$ for the sentence S:

$$\arg\max_{G\in\mathcal{G}_S} score(G) \tag{3.6}$$

This is the *maximum spanning tree* (MST) on directed graphs (more precisely, *multidigraphs* - directed multigraphs because arcs with different labels between two nodes are different arcs), a well-studied problem in graph theory. The score of a tree can be decomposed into the score of smaller components:

$$score(G) = \sum_{\psi\in G} score(\psi) \tag{3.7}$$

where ψ denotes a subgraph in G. Without loss of generality, we assume the tree score is the sum of all score factors, but it can be also defined as a product if each score factor represents a probability.

Arc-factored models Probably, the most straightforward way to decompose a tree is into individual arcs. Following this, $\psi = (w_i, r, w_j)$ and the score of a dependency tree becomes:

$$score(G = (V, A)) = \sum_{(w_i, r, w_j) \in A} score(w_i, r, w_j) \tag{3.8}$$

This type of models is known as the *arc-factored* models and also as *first-order* models because it utilizes the first-order relation between nodes in a tree. The works of McDonald et al. (2005a) and McDonald et al. (2005b) are among the first to formalize dependency parsing as finding the MST in a directed graph with arc-factored models.

Parsing with arc-factored models Without any restriction, the search space for the MST includes both projective and non-projective dependency trees, so finding the MST is equivalent to non-projective parsing. MSTs on directed graphs can be solved with the Chu-Liu-Edmonds (Chu and Liu, 1965; Edmonds, 1967) algorithm that runs in $O(n^3)$ time, but the run time can be reduced to $O(n^2)$ with a more efficient implementation in Tarjan (1977). The algorithm involves both greedy and recursive procedures:

- Given a graph G, we first try to construct the MST by *greedily* selecting the highest incoming arc for each node into a subgraph G_1. If the process creates a tree, the algorithm has success and returns it as the MST.
- If G_1 is not a tree, then there must be a cycle G_C. This cycle is identified and is contracted into a new node. Arcs that have one end in the cycle have their weights recalculated.
- The algorithm *recursively* finds the MST in the newly contracted graph G'.
- The MST G_2 on G' has exactly one incoming arc to the contracted node, and this arc helps us to remove one edge from the cycle G_C, thus breaks it. Arcs in G_2 are replaced with the corresponding ones from G. Combining the remaining arcs in the G_C and G_2 results in the MST for the original graph.

The Chu-Liu-Edmonds algorithm operates on directed graphs, which corresponds to unlabeled parsing. Fortunately, labeled parsing can be reduced to unlabeled parsing with a simple procedure. Given a multidigraph $G_S = (V_S, A_S)$ with arc weights λ, we construct a graph $G'_S = (V'_S, A'_S)$ with arc weights λ' such that:

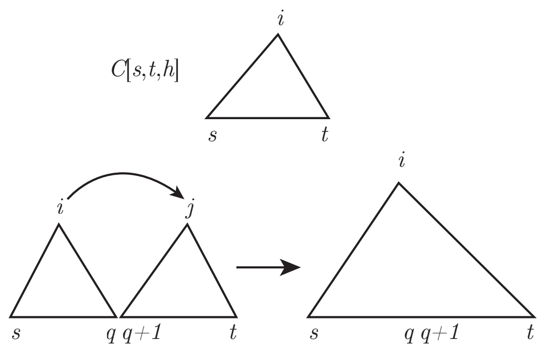


Figure 3.7: CKY parsing algorithm

- $V'_S = V_S$
- $A'_S = \{(w_i, w_j) \mid \exists (w_i, r, w_j) \in A_S\}$
- $\lambda'(w_i, w_j) = \max_r \lambda'(w_i, r, w_j)$

It can be proven that the MST $G' = (V', A')$ on G'_S is equivalent to the MST $G = (V, A)$ on G_S if we label each arc (w_i, w_j) in V' with $r = \arg\max_{r'} \lambda(w_i, r', w_j)$ (see Kübler et al. (2009) for the proof). The process has to go through $O(n^2)$ unlabeled edges, each has $|R|$ options for labels, thus its running time is $O(|R|n^2)$. Because of that, the Chu-Li-Edmonds algorithm for labeled dependency parsing also runs in $O(|R|n^2)$ time.

Unlike unrestricted MSTs, projective MSTs are handled with bottom-up, dynamic programming chart parsing algorithms inspired by the Cocke-Kasami-Younger (CKY) algorithm for constituency parsing. Basically, the problem is broken down into smaller sub-problems: to find the projective tree with the highest score that spans the nodes from s to t and roots at a node w_h, $s \leq h \leq t$. Its score is stored in table C, at entry $C[s, t, h]$. For presentation purpose, the tree is illustrated as in figure 3.7 by a triangle with 3 vertices s, h, t. The projective MST is the tree associated with $C[0, n, 0]$. Two adjacent trees at spans (s, q) and $(q + 1, t)$ can be combined by adding an arc connecting their two heads (figure 3.7), and the score of the new tree is equal to the sum of the two subtree scores, plus the score of the new edge. Thus, the algorithm uses the recursive procedure below:

$$C[s,t,i] = \max_{s \leq q < t, s \leq j \leq t} \begin{cases} C[s,q,i] + C[q+1,t,j] + \lambda(w_i, w_j) & i < j \text{ (add a right arc)} \\ C[s,q,j] + C[q+1,t,i] + \lambda(w_i, w_j) & j < i \text{ (add a left arc)} \end{cases} \tag{3.9}$$

To fill each entry, the algorithm considers $O(n^2)$ possibilities of (q, t). Thus, the naive CKY algorithm runs in $O(n^5)$ time to fill $O(n^3)$ entries for unlabeled parsing.

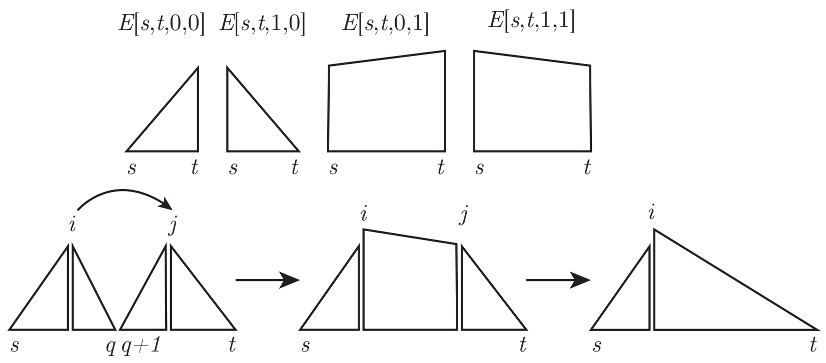


Figure 3.8: Eisner parsing algorithm

Labels can be incorporated directly into the algorithm with time complexity $O(|R|n^5)$. Alternatively, if the problem is first reduced to unlabeled parsing before applying the algorithm, the total runtime is $O(n^5 + |R|n^2)$. Its memory requirement is $O(n^3)$.

The runtime of the naive algorithm is not feasible in practice, but it can be reduced to $O(n^3)$ by a variant proposed in Eisner (1996). Eisner (1996) observes that a node w_i can collect its left and right dependencies independently, and the score of the whole tree at w_i can be combined at a later stage. Thus, the new sub-problem only takes into account a tree spanning from s to t that rooted at either s or t, dropping one index and requiring only $O(n^2)$ space. The dynamic programming table now has a form of $E[s, t, d, c]$ that stores the highest scored tree in the span (s, t), with its head is either w_t (when $d = 0$) or w_s (when $d = 1$). The index c indicates if the subgraph still needs to be combined with its left or right subgraph (when $c = 1$) or not (when $c = 0$). Figure 3.8 displays all possible forms of subgraphs in the Eisner algorithm and the combination steps. The score of the MST is at $E[0, n, 1, 0]$.

Learning arc-factored models Similar to transition-based parsing, arc-factored approaches define the score of an arc as the dot product between a weight vector $\mathbf{w}$ and a feature vector of the arc $\mathbf{f}(w_i, w_j)$:

$$score(w_i, w_j) = \mathbf{w} \cdot \mathbf{f}(w_i, w_j) \tag{3.10}$$

If labels are also taken into account, a relation-specific weight vector $\mathbf{w}_r$ is used to compute the score of an arc with label r:

$$score(w_i, r, w_j) = \mathbf{w}_r \cdot \mathbf{f}(w_i, r, w_j) \tag{3.11}$$

but since labeled parsing can be easily extended to, or reduced to unlabeled parsing, for simplicity, we only consider the unlabeled problem. Structured learning is also applied to learn the parameters of graph-based models as in transition-based parsing

Algorithm 7: Generic averaged perceptron algorithm for graph-based parsers

```
input  : set of training examples $\{(x_i, y_i)\}_{i=1}^N$
         $x_i$ is a sentence and $y_i$ is the dependency tree of $x_i$
output : weight vector $\mathbf{w}$
1  $\mathbf{w}^0 \leftarrow 0$, $\mathbf{v} \leftarrow 0$
2  for $t \in 1..T$ do
3      foreach training example $(x_i, y_i)$ do
4          $y_p \leftarrow \arg\max_{y \in \mathcal{G}_{x_i}} score(x_i, y, \mathbf{w}^{(t)})$
5          if $y_p \neq y$ then
6              $\mathbf{w}^{(t+1)} \leftarrow \textsc{Update}(\mathbf{w}^{(t)}, y, y_p)$
7          else
8              $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)}$
9          end
10         $\mathbf{v} \leftarrow \mathbf{v} + \mathbf{w}^{(t+1)}$
11     end
12     $\mathbf{w} \leftarrow \mathbf{v}/(T \times N)$
13 end
```

(section 3.1.3). In fact, the first application of the perceptron algorithm (Collins, 2002) to transition-based parsing (Zhang and Clark, 2008) was inspired by its usage for arc-factored models in McDonald et al. (2005a). Algorithm 7 is a generic online training algorithm for graph-based parsing models. It shares many similarities with the online training algorithm for transition-based parsing (algorithm 3). Here, y_p is the highest scored tree in the set of all possible trees $\mathcal{G}_{x_i}$ of the sentence x_i. In arc-factored models:

$$y_p = \arg\max_{y \in \mathcal{G}_{x_i}} \sum_{(i,j) \in y_i} \mathbf{w}^{(t)} \cdot \mathbf{f}(i,j) \tag{3.12}$$

and the $\arg\max$ can be solved with the Chu-Liu-Edmond or Eisner algorithm. The final weight vector $\mathbf{w}$ is the average of the weight vector in each iteration, which is known to reduce overfitting (Collins, 2002) (thus, is known as the averaged perceptron). If using the perceptron algorithm, the update procedure (line 6) is:

$$\mathbf{w}^{(t+1} = \mathbf{w}^{(t)} + \sum_{(i,j) \in y} \mathbf{f}(i,j) - \sum_{(i,j) \in y_p} \mathbf{f}(i,j) \tag{3.13}$$

McDonald et al. (2005a) and McDonald et al. (2005b) use the Margin Infused Relaxed Algorithm (MIRA) (Crammer and Singer, 2003), a large-margin version of the perceptron algorithm. MIRA is formalized for graph-based parsing as:

$$\begin{aligned}
&\min \|\mathbf{w}^{(t+1)} - \mathbf{w}^{(t)}\| \\
&\text{s.t. } score(x_i, y_i, \mathbf{w}^{(t+1)}) - score(x_i, y', \mathbf{w}^{(t+1)}) \geq L(y_i, y') \\
&\forall y' \in \mathcal{G}_{x_i}
\end{aligned} \tag{3.14}$$

$L(y, y')$ is the loss of the tree y' with regard to the correct tree y, for example, the number of arcs with incorrect heads in y'. The original MIRA uses all trees in the possible tree space $\mathcal{G}_{x_i}$ of the sentence x_i to update the weight. It is impractical for dependency parsing because the number of possible trees is exponential. McDonald et al. (2005a) approximate this with the top k trees, acquired by modifying the Eisner algorithm. The authors report that this *k-best MIRA* performs slightly better than the averaged perceptron. However, the gain of the k-best MIRA is very minor in comparison to the *single-best MIRA* (when $k = 1$) while the training time is less efficient. Besides structured learning, there are other probabilistic frameworks to learn the parameters of arc-factored models that rely on computing the *partition function* (the sum of tree scores) and arc expectation, including log-linear (Koo et al., 2007) and generative models (Klein and Manning, 2004).

For each arc, the atomic features of arc-factored models are the word form and POS tag of the head, the dependent, and the surrounding nodes. The neighbor nodes of interest are: two nodes surrounding the head, two nodes surrounding the dependent, and the feature function also uses the POS tags of all nodes between the head and the dependent. See McDonald et al. (2005a) for the full set of atomic and combined features.

Higher-order parsing At first glance, arc-factored models are very appealing: They are simple and can do exact decoding based on graph theory which is practically efficient and can be optimized globally. Nevertheless, its naive assumption about arc independence becomes its Achilles heel, simply because the assumption does not hold in syntax. A well-known error of first-order models is that they often predict two subjects. Therefore, there is a strong desire to incorporate higher relations in the existing framework of arc-factored parsing. Higher-order relations in dependency parsing are illustrated in figure 3.9. They can be classified into two types: vertical and horizontal. The vertical neighborhood of an arc (w_i, w_j) includes all nodes on the path from the root that passes through (w_i, w_j), i.e., parent, grandparent, child, grandchild, etc., relations. The horizontal neighborhood of the arc (w_i, w_j) includes all children of the node w_i, i.e., sibling relations.

Second-order models for projective dependency parsing are presented in McDonald and Pereira (2006) (sibling) and Carreras (2007) (sibling and grandparent). For

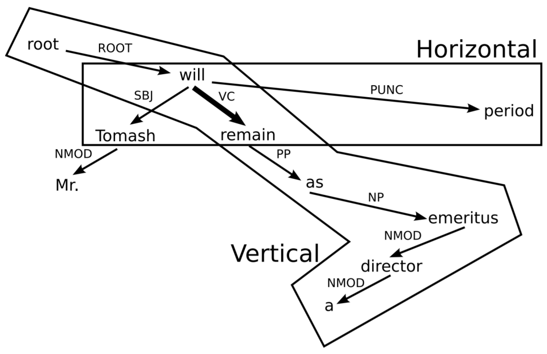


Figure 3.9: Horizontal and vertical neighborhoods for the arc (will, VC, remain) (reproduced from McDonald and Satta (2007))

instance, the score of a dependency tree G according to the second-order model from Carreras (2007)[1] is calculated as:

$$score(G) = \sum_{(h,m,c_h,c_{mi},c_{mo})\in G} score(h,m,c_h,c_{mi},c_{mo}) \tag{3.15}$$

$$\begin{aligned} score(h,m,c_h,c_{mi},c_{mo}) = \mathbf{w}\cdot\mathbf{f}_1(h,m) + \mathbf{w}_h\cdot\mathbf{f}_2(h,m,c_h) \\ + \mathbf{w}_{mi}\cdot\mathbf{f}_1(h,m,c_{mi}) + \mathbf{w}_{mo}\cdot\mathbf{f}_2(h,m,c_{mo}) \end{aligned} \tag{3.16}$$

where c_h is the dependent of h that is closest to m, c_{mi} is the dependent of m inside $[h...m]$ that is furthest from m, and c_{mo} is the dependent of m outside $[h...m]$ that is furthest from m. $\mathbf{f}_1(\cdot)$ and $\mathbf{f}_2(\cdot)$ are first-order and second-order feature functions, respectively. The Eisner algorithm can be extended to parse an arbitrary m-th order model in polynomial time ($O(n^3)$ in McDonald and Pereira (2006) and $O(n^4)$ in Carreras (2007)). Even so, $O(n^4)$ time complexity is too slow to apply the algorithm directly in real applications. In the case of non-projective parsing, unfortunately, higher-order models are intractable. Dependency parsing with either vertical or horizontal relations is shown to be NP-hard by a reduction from 3-dimensional matching (McDonald and Pereira, 2006; McDonald and Satta, 2007).

The attempt to incorporate higher-order information into graph-based parsing leads to research in approximation algorithms. Inspired by the fact that many non-projective trees only have a few non-projective arcs even in languages with flexible word order like Czech, McDonald et al. (2005b) propose simple approximate non-projective parsing with a higher-order model. First, a second-order projective parser is used to find the best projective tree with exact decoding. From that, the approximation works by changing the head of one node in the tree at a time, as long as it is a

[1]We omit labels from the original formula for simplicity.

valid tree and the overall score improves. An arc (w_k, w_j) is chosen to replace an arc (w_i, w_j) if the gain in scores is the biggest:

$$w_k = \arg\max_{i'} score(w_{i'}, w_j) - score(w_i, w_j) \tag{3.17}$$

The number of changes is bounded by a fixed number M, so the runtime of the process is $O(Mn^2)$, resulting in $O(Mn^2 + n^3)$ time complexity for approximate second-order non-projective parsing. This technique improves parsing accuracy for Czech and Danish, showing that the higher-order model can make up for the heuristics in approximate parsing. Following up, the work of Bohnet (2010) addresses the runtime bottleneck of graph-based parsers, which is feature extraction. By using feature hashing and parallelization, a parser combining the second model of Carreras (2007) and the non-projective parsing approximation above runs 3.5 times faster.

Furthermore, higher-order information can be injected beyond parsing, as in post-processing. Hall (2007) uses the k-best MST (without the projectivity restriction) algorithm (Camerini et al., 1980) to extract top k trees from an arc-factored model. The trees are then reranked with higher-order features. Hall (2007) reports minor improvements over the base parser for 8 languages. In chapter 7, we compare the performance of 3 different neural reranking models in experiments with 3 languages.

Two-staged parsers Ideally, unlabeled parsing and labeling are done jointly so that the joint system can take advantage of shared knowledge. With graph-based dependency parsing, however, adding labels to the model may cause over-parameterization, which makes the optimization process less effective. On the other hand, because the higher-order models are either intractable or impractical, labeling is also restricted by the set of local features, resulting in errors like predicting two subjects for the same clause. McDonald et al. (2006) train a separate labeler, formulating the task as a sequence labeling problem, using Conditional Random Fields (CRFs). Their two-stage parser showed significant improvements over the average performance of joint parsers. In chapter 5, we follow this approach and implement three neural grammatical function labelers with history, and show that they are all better than the baseline labeler that predicts the label of each arc independently.

3.1.5 Comparing Transition-Based and Graph-Based Parsing

In the previous sections, we have learned about two different approaches to model dependency parsing: transition-based and graph-based. In order to contrast these two approaches, we compare them based on these properties (summarized in table 3.2):

	Transition-based	Graph-based
Parameterization	Transitions	Subgraphs
Features	Non-local	Local (1st, 2nd.. order)
Learning	Local (SVMs) or global (perceptron)	Global (MIRA)
Inference	Greedy or beam search	Near exhaustive (exact)

Table 3.2: Comparing transition-based and graph-based dependency parsing

- **Parameterization:** Given a sentence, a transition-based parser models transitions of an abstract machine, where each state represents a partially parsed dependency tree of the sentence. A graph-based parser parameterizes subgraphs of the complete directed graph of the input sentence.
- **Inference:** A transition-based parser can be designed to return one best transition at a current configuration (*greedy*) or it can approximate global decoding using beam search. Conversely, a graph-based parser yields the exact best tree by using graph algorithms to find the MST.
- **Features:** A transition-based parser can easily incorporate arbitrary features without changing its decoding process, whereas a graph-based parser is forced to restrict the feature scope to make exhaustive inference tractable and practical.
- **Learning:** A transition-based parser can be trained locally (to update the parameters based on an incorrect transition) or globally (to update the parameters based on an incorrect sequence of transitions). A graph-based parser is always trained globally.

Before the addition of beam search or global training to transition-based parsing, the combination of local training and greedy decoding makes transition-based parsers very attractive because they are very fast while being reasonably accurate. Thus, early works like McDonald and Nivre (2007) brand transition-based approaches as *local* and *greedy* to contrast them with *global* and *exhaustive* graph-based approaches. Despite their different nature, MaltParser (Nivre et al., 2006a) (a transition-based parser) and MSTParser (McDonald and Pereira, 2006) (a second-order graph-based parser) achieve very similar accuracies on the 13 languages included in the CoNLL-X Shared Task. By analyzing their error types, McDonald and Nivre (2007) demonstrate a trade-off between global training and exhaustive decoding on the one hand, and rich feature representations on the other. Local training and greedy inference make MaltParser prone to error propagation, thus performing worse on longer sentences and longer dependency arcs. In contrast, its rich feature representation benefits frequent arc types and short dependencies, such as subjects and objects. Later, Zhang and Clark (2008) consider the effect of beam search and global training on dependency parsing and show that ZPar (Zhang and Nivre, 2011) (a transition-based parser with non-local features, global training, and beam search decoding) surpasses both MaltParser and MSTParser.

The only question left unanswered is whether the parameterization, or the way the two approaches define the parsing problem, has any effect on parsing results. Although there is no direct answer, the improvement gained when combining these two approaches in one system suggests that the answer to the question is the parameterization matters. Zhang and Clark (2008) augment the scoring function of a transition-based parser by combining it with the scoring function of a graph-based parser. When decoding with beam search, each hypothesis h in the beam is assigned a score $score_T(h)$ by the original transition-based model:

$$score_T(h) = score_T(C_{0,m}) \tag{3.18}$$

where $C_{0,m}$ is the transition sequence leading to h. Moreover, h also corresponds to a partial parsed tree A, so it can also be scored with a graph-based model:

$$score_G(h) = score_G(A) \tag{3.19}$$

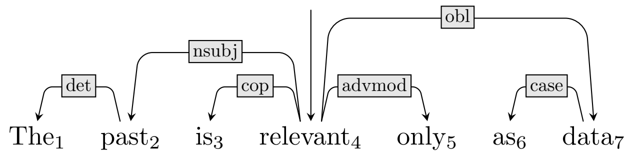
The combination score is simply a summation:

$$score(h) = score_T(h) + score_G(h) \tag{3.20}$$

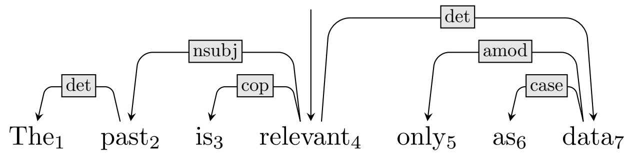
Bohnet and Kuhn (2012) have basically the same approach, but they take an extra step to recalculate the score of each hypothesis in the beam when a new graph factor becomes available.

3.1.6 Evaluating Dependency Parsers

The most widely used metrics in dependency parsing evaluation are *attachment scores*. *Unlabeled attachment score* (UAS) is the percentage of words that are assigned the correct head. *Labeled attachment score* (LAS) is the percentage of words that are assigned the correct head and the correct label. An example of UAS and LAS calculation is shown in figure 3.10. Sometimes punctuation is excluded from the list of words when calculating attachment scores.



(a) gold tree



(b) predicted tree

word	has correct head	has correct head & label
1	1	1
2	1	1
3	1	1
4	1	1
5	0	0
6	1	1
7	1	0
Total		
7	6	5

Figure 3.10: An example of evaluation metrics in dependency parsing. The UAS and LAS of the predicted tree (b) with respect to the gold tree (a) are 6/7 and 5/7 respectively.

3.2 Neural Approaches in Dependency Parsing

The term *artificial neural networks* (or simply known today as *neural networks*) has a history dating back to the 1940s (also see chapter 2.1 for an introduction of neural networks); however, the technique has only become popular only in the last two decades. The first successful neural systems in computer vision use algorithms proposed in the 1980s which were too computationally costly back then, and are only practically feasible with recent hardware developments. Following this trend, NLP research has also tried to solve long-time problems with neural network methods. For the rest of this section, we first travel back to the point where the first neural parser was introduced and see how neural networks have changed the way parsers are built, or NLP research in general.

3.2.1 The First Neural Dependency Parser

Chen and Manning (2014) integrate a neural network classifier in a transition-based dependency parser, known today as the first neural dependency parser. At its core, the parser (CM parser, see figure 3.11) is a local, greedy transition-based parser (section 3.1.3) similar to MaltParser (Nivre et al., 2006a) but with two modifications. First, the sparse and discrete encoding (one-hot vectors) of input features was replaced with continuous and dense vectors (embeddings) (see section 2.2.2). Word embeddings are initialized with vectors that are acquired by training a language model on external, unlabeled data. Chen and Manning (2014) are also the pioneers

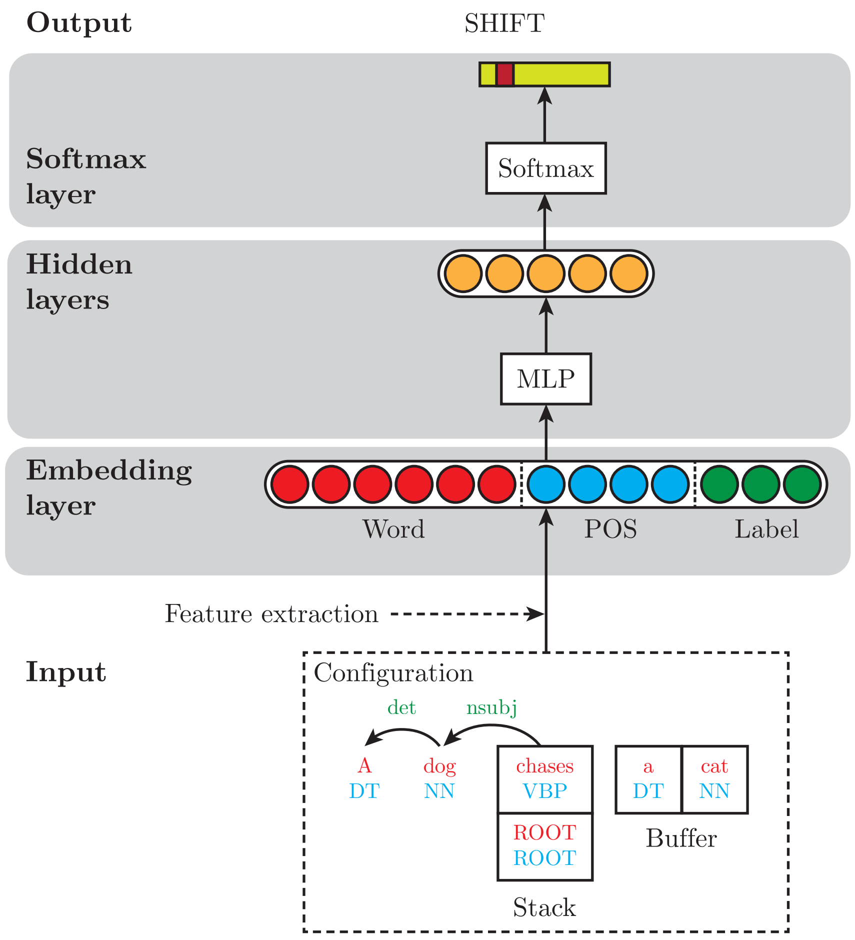


Figure 3.11: The architecture of Chen and Manning (2014)'s parser, a local, greedy neural transition-based parser

of representing not just words, but arbitrary features with embeddings, like POS and dependency labels. Second, the authors substitute the linear SVM classifier with a non-linear one hidden layer feed-forward neural network (section 2.1.2).

Representing core features with embeddings together with using a non-linear classifier eliminates the need for hand-crafting *feature combinations* because the classifier can automatically *learn* to combine *core* features during training. However, Chen and Manning (2014) still rely on previous works to determine the set of core features. Using the arc-standard oracle, their feature set contains 18 positional elements, e.g., the top 3 words on the stack and the buffer, the first and second leftmost/rightmost children of the top two words on the stack, etc.

The modifications help the first neural dependency parser gain improvements of 2% UAS and LAS on English data over its non-neural counterpart. Most of the improvement comes from the structural changes (the pre-trained embeddings contribute 0.7%), which confirms the effectiveness of neural networks and embeddings over linear models. Using non-linear models with dense embeddings to replace linear models with sparse encoding is a common recipe to turn conventional NLP systems into neural ones.

3.2.2 Back to Global Optimization

Although having great success with neural networks, the CM parser is still inferior to parsing models that are trained for structured prediction. Thus, there is a line of work that focuses on incorporating global optimization into neural parsers. Weiss et al. (2015) pre-train the CM parser and concatenate the representations from its output and hidden layers as feature vectors to train a perceptron layer (similar to algorithm 4). The authors achieve faster and better results with pre-training in comparison to full back propagation. The structured perceptron training gains 0.8% improvement in both UAS and LAS, achieving better results than all non-neural systems on English data.

Zhou et al. (2015) also experiment with global training using ranking loss. The reason why full back propagation fails with neural models, according to the authors, is because the action space shares many parameters (unlike linear models), so increasing the likelihood of a gold transition over an incorrect one may also change the likelihood of incorrect transitions. Instead, they propose to combine a local, greedy neural parser with a probabilistic structured prediction model that maximizes the *probability* of the gold transition sequence, using contrastive learning to approximate the loss. Their global model achieves 1.8% higher UAS than the greedy baseline, but the beam size used is very large (100).

Andor et al. (2016) replace the perceptron layer in Weiss et al. (2015) with a CRF layer and perform full back propagation. They outperform both Weiss et al. (2015) (+0.6%) and Zhou et al. (2015) despite using a much smaller beam size (32). The authors also show that the difference between the CRF loss and the perceptron loss is negligible when training the top layer (CRFs or perceptron) only, but full training with CRFs converges 4 times faster and yields better results.

3.2.3 Unbounded Features with Stack LSTMs

Another extension to the work of Chen and Manning (2014) tackles the hand-crafted core features. Long Short Term Memory networks (LSTMS) (see section 2.1.3) is a variation of recurrent neural networks (RNNs) designed to overcome the vanishing gradient problem (Hochreiter and Schmidhuber, 1997). LSTMs, or RNNs in general, are used to capture long-range dependencies in a sequence of arbitrary length. The last hidden state of a sequence passed through an LSTM unit contains information on

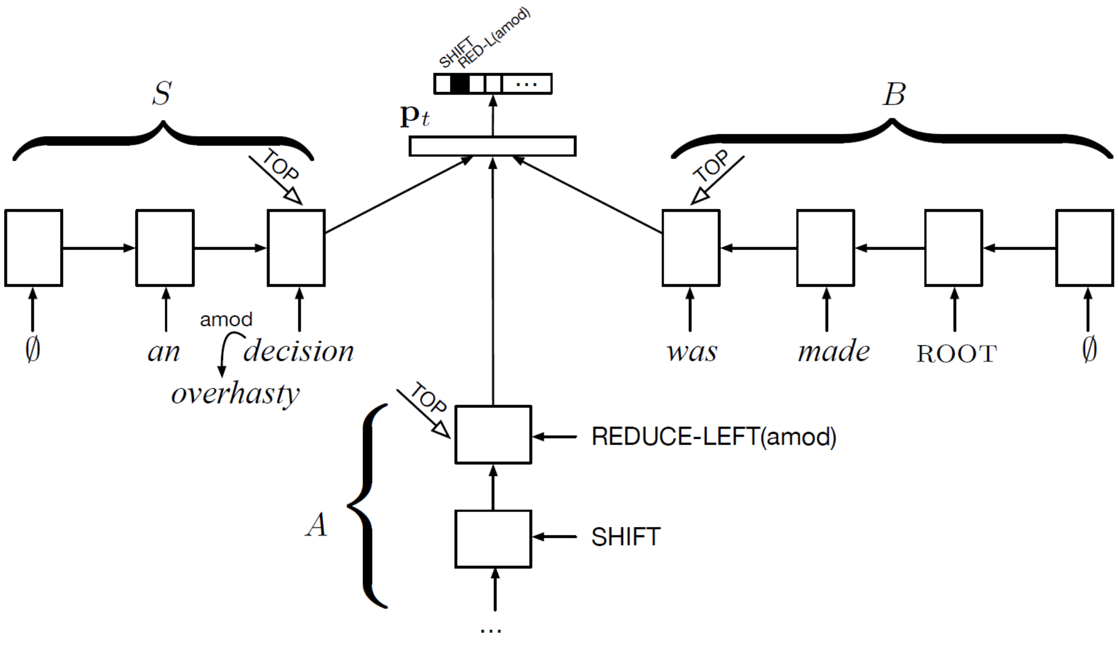


Figure 3.12: A parsing configuration with three stack LSTMs (reproduced from Dyer et al. (2015))

all elements in that sequence. In Dyer et al. (2015), LSTMs are used to automatically summarize the information of three components in a transition-based dependency parser: the stack, the buffer, and the additional transition history. Treating all three components as stack structures, the authors adjust the vanilla LSTMs with stack operations, resulting in *stack LSTMs*. The concatenation of the last hidden states of the stack, the buffer, and the history (figure 3.12) takes the place of the embedding layers with manual feature extraction in the CM parser. The stack LSTM parser improves over the CM parser but is far behind the neural parsers with structured prediction.

3.2.4 Word Representations with Bidirectional LSTMs

If we treat the input sentence as a sequence of words and apply an RNN on it, the hidden state of a word computed by the RNN represents the information from the previous words up to and including the current word. However, sometimes information of the following words is also important, as for parsing. In a similar fashion, an RNN applied in the *reverse* direction can capture the information from the future up to and including the current word. Combining the hidden states from both RNNs results in the representation of a word and the context surrounding it. The two RNNs are together called a *bidirectional* RNN. Bidirectional RNNs are first proposed in Schuster and Paliwal (1997), in which they are applied to speech processing. The first use of bidirectional RNNs in NLP traces back to their application in opinion mining by İrsoy and Cardie (2014).

In other words, bidirectional RNNs *encode* a word in a sentence with respect to its context. The term *encode* seems to be coined by the usage of RNNs in *sequence-to-sequence* models (Cho et al., 2014; Sutskever et al., 2014; Bahdanau et al., 2015). A sequence-to-sequence problem has sequences as both input and output, and with possibly different lengths, like machine translation or summarization. Most of the neural approaches for sequence-to-sequence problems use the *encoder-decoder* architecture where an *encoder* reads and encodes the input sequence and a *decoder* generates the output sequence based on the encoded information. The standard practice of using bidirectional RNNs to encode sentences appears to originate from Bahdanau et al. (2015), where the encoder consists of *multilayer bidirectional LSTMs* (multiple layers of stacked bidirectional LSTMs), and the encoded representation of a word is the concatenation of the forward and backward hidden states of the last BiLSTM layer.

It is worth mentioning the first application of bidirectional LSTM encoders for syntactic parsing in Vinyals et al. (2015). The authors directly use the *sequence-to-sequence model with attention* (Bahdanau et al., 2015) for constituency parsing by linearizing output trees as sequences with brackets. Following this, two independent works by Kiperwasser and Goldberg (2016b) and Cross and Huang (2016) propose to replace the hand-crafted feature set in dependency parsing with automatically learned features using the output of bidirectional LSTMs.

Kiperwasser and Goldberg (2016b) employ a greedy transition-based parser with a dynamic arc-hybrid oracle. Words in a sentence are encoded by multilayer bidirectional LSTMs. Similar to Chen and Manning (2014), they use a multilayer perceptron (MLP) to score the next transitions. They test two different feature sets for parsing. The *simple features* consist of the BiLSTM vectors of the top 3 items on the stack and the first item on the buffer (4 feature vectors, table 3.3). Additional words from the buffer are not needed because the bidirectional LSTM representation of the first item on the buffer is expected to capture them. The *extended feature set* additionally includes information on the already built tree, resulting in a set of 11 bidirectional LSTM feature vectors in total (11 feature vectors, table 3.3).

For graph-based parsing, Kiperwasser and Goldberg (2016b) use two MLPs to score the edge and the label between two words. Inputs to the MLPs are the bidirectional LSTM representations of the head and the dependent. The graph-based parser is trained with a margin-based hinge loss to maximize the margin between the score of the correct tree and the highest scored incorrect one.

Cross and Huang (2016) introduce a feature set identical to the one of Kiperwasser and Goldberg (2016b) for a greedy transition-based parser with arc-standard oracle. They also apply the same method for greedy transition-based constituency parsing and outperform the beam search approach in Vinyals et al. (2015) by a large margin (89.95 vs. 88.3 F score) when trained on the Penn Treebank without ensemble and additional data. The simple approaches by Kiperwasser and Goldberg (2016b) and Cross and Huang (2016) surpass the CM parser and are also better than the stack-based parser of Dyer et al. (2015) for transition-based parsing (table 3.3). Even the minimalistic graph-based parser from Kiperwasser and Goldberg (2016b) has comparable results to Dyer et al. (2015) (table 3.3).

3.2.5 More Powerful, but Simpler Parsers

In contrast to the pre-neural era, neural dependency parsers have become less complicated and while achieving high results without the need to incorporate beam search and global training. In 2017, Zhang et al. (2017) take simplicity to a further step by proposing a neural model that learns to predict the head of a word, which is promoted as *parsing as head selection*. At its core, the model of Zhang et al. (2017) is a graph-based dependency parser with bidirectional LSTM features similar to Kiperwasser and Goldberg (2016b). However, unlike Kiperwasser and Goldberg (2016b) who train their parser globally, Zhang et al. (2017) backpropagate the head prediction of each word *independently*, resulting in a *locally* trained graph-based parser. At test time, since greedy inference does not guarantee that the output is a sound tree, the Eisner algorithm is applied to build projective trees and the Chu-Liu-Edmond algorithm to non-projective ones (see section 3.1.4). In fact, these algorithms are only used as a post-processing step when the model outputs non-trees. Surprisingly, post-processing does not contribute much to the final scores (less than 0.1%) in experiments with English, German and Czech, confirming the potential of the greedy model. In experiments with the English Penn Treebank, Zhang et al. (2017) further improve the parsing performance in comparison to Kiperwasser and Goldberg (2016b), and their results are only 0.5% lower than the globally optimized parser from Andor et al. (2016) (table 3.3).

In a work independent from Zhang et al. (2017), Dozat and Manning (2017) propose a similar model that uses *biaffine attention* instead of MLPs (Kiperwasser and Goldberg, 2016b; Zhang et al., 2017) to predict the head and the grammatical function of each word. A biaffine transformation of two vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ is simply a

System	UAS	LAS	Method
Transition-based			
Chen and Manning (2014)	91.8	89.6	greedy
Weiss et al. (2015)			
Greedy	93.19	91.18	beam size 1
Perceptron	93.99	92.05	beam size 8
Andor et al. (2016)	94.61	92.79	CRFs, beam size 32
Dyer et al. (2015)	93.2	90.9	greedy
Kiperwasser and Goldberg (2016b)			greedy, dynamic oracle
Simple features	93.6	91.5	4 vectors
Extended features	93.9	91.9	11 vectors
Cross and Huang (2016)	93.42	91.36	greedy
Graph-based			
Kiperwasser and Goldberg (2016b)	93.0	90.9	1st order
Zhang et al. (2017)	94.1	91.9	1st order
Dozat and Manning (2017)	95.74	94.08	1st order

Table 3.3: Parsing results of various neural parsers on the English PTB test set with Stanford Dependencies. Note that the different systems may use different versions of the Stanford converter, which means that the results are not necessarily comparable.

combination of a bilinear and an affine transformations:

$$\underbrace{\mathbf{x}_1^\top \mathbf{W} \mathbf{x}_2}_{\text{bilinear}} + \underbrace{\mathbf{x}_1^\top \mathbf{w}_1 + \mathbf{x}_2^\top \mathbf{w}_2}_{\text{affine}} \tag{3.21}$$

where $\mathbf{W}$, $\mathbf{w}_1$, and $\mathbf{w}_2$ are parameters. Despite its simplicity, the effectiveness of biaffine classifiers together with carefully optimized implementation and hyper-parameter selection helps the parser of Dozat and Manning (2017) achieve very high parsing performance, surpassing the similar approach in Zhang et al. (2017) by 2% LAS, and even outperforming the globally trained parser of Andor et al. (2016) (table 3.3).

3.2.6 Summary and Further Approaches

Recent work in neural dependency parsing has slowly begun to replace components of traditional dependency parsers with neural ones. First, sparse and discrete input features have been replaced by embeddings, and then linear classifiers have been substituted for MLPs. After that, manual feature templates give up their places for global, unbounded features encoded by RNNs. Up to now, the parameterization (i.e., transition-based or graph-based) of dependency parsers has almost remained the same, but the power of neural networks has enabled a large number of simpler and greedy models, some even outperform global ones. However, as in other neural network applications, there is no clear line between the contribution of neural architectures on the one hand and on the other the impact of optimization on the final results. As in the case of Dozat and Manning (2017), effective optimization plays a great role for the success of a neural system.

An exhaustive description of all current approaches in neural dependency parsing would exceed the scope of this thesis. Besides the prominent trends above, some works incorporate neural networks in probabilistic models (Ma and Hovy, 2017). Higher-order neural graph-based dependency parsers are introduced in (Ji et al., 2019; Zhang et al., 2020). In Kiperwasser and Goldberg (2016a), LSTMs are used to represent tree forests in easy-first transition-based parsing. Novel parsing architectures with pointer networks are proposed in Ma et al. (2018) and Fernández-González and Gómez-Rodríguez (2019).

Pre-trained neural language models, which provide word representations that are *context-sensitive*, can be used to replace conventional pre-trained word embeddings and achieve state-of-the-art results in many tasks, including dependency parsing (Peters et al., 2018) (see section 2.2.4). This is also the technique we employ to improve PP attachment disambiguation in chapter 6. Besides using pre-trained language models to generate input embeddings only, recent approaches replace most of the components in conventional neural dependency parsers with a pre-trained language model and fine-tune the language model with dependency parsing data. Kondratyuk and Straka (2019) replace the bidirectional LSTM representations in the biaffine parser (Dozat and Manning, 2017) with a weighted sum of representations at all layers of BERT (Devlin et al., 2019). They then fine-tune the cased multilingual BERT model (trained on Wikipedia for 104 languages) in a multitask setting on the UD corpus which has 75 languages, and the multilingual model achieves state-of-the-art results on POS tagging and dependency parsing without any language- or task-specific components. Glavaš and Vulić (2020) further remove other components and only keep the biaffine classifiers on top of the pre-trained language model in their dependency parser.

Other methods to improve dependency parsing with neural networks include character-based and subword embeddings (Ballesteros et al., 2015) (see section 2.2.2) and multi-task learning (Hashimoto et al., 2017).

CHAPTER 4

Word and Morphological Level: The Unknown Word Problem

In this chapter, we look into the first structural level of linguistics: the **(sub)word and morphological** level. At this level, the main challenge for parsing morphologically rich languages (MRLs) is the high proportion of unknown words in the data, due to the high number of different inflected forms. In some languages, this problem is made even worse by *compounding*, a highly productive word formation process. Thus, handling unknown words is crucial for parsing MRLs and especially for German where compounding is a frequent phenomenon.

While word embeddings are a promising way to learn a general representation that captures syntactic and semantic properties of a word, they have not fully solved the sparse data problem. Recent studies are exploring representations at the subword level that can provide information even for rare and unseen words. Well-known examples are character and character-n-gram-based embeddings (Sperr et al., 2013; dos Santos and Zadrozny, 2014; Ling et al., 2015b; Vania and Lopez, 2017), morphological embeddings (Luong et al., 2013; Botha and Blunsom, 2014; Cotterell and Schütze, 2015; Cao and Rei, 2016), or byte embeddings (Plank et al., 2016; Gillick et al., 2016).

Ballesteros et al. (2015) were the first to integrate character-based embeddings into a syntactic parser and compared the effect for different languages with different levels of morphological richness. They showed that replacing word embeddings with character-based embeddings can be useful, especially for parsing agglutinative languages. Since then, character-based embeddings have become an ingredient in many parsing systems.

Other work has addressed the compounding problem on the level of word embeddings. Dima et al. have tried to model compound compositionality for English (Dima and Hinrichs, 2015) and German (Dima, 2015). However, experiments were on the

Threshold	`en`	`de`
1	13.17	37.18
2	19.48	48.78
3	24.39	55.59
4	28.36	60.51
5	31.90	64.12

Table 4.1: The percentage of unknown words in the *test* data set with respect to different levels of cutoff thresholds in the *training* data. A threshold of 1 means no words in the training data are discarded.

semantic level, and the compounds were restricted to two components only. To the best of our knowledge, nobody has tried compound embeddings to tackle the unknown word problem in statistical parsing.

4.1 The Problem with Compounds

Compounds are words that include more than one stem. In some languages (e.g. English), the individual components are separated by spaces, while in other languages, such as German, they are merged into a new word form. Compounding is highly productive and thus, in languages like German, a major source of new, unseen words. Take, for example, the German compound *Verbraucherschutzgesetz* (consumer protection law). While all three parts are reasonably frequent and thus have a good chance of being included in a sufficiently large data set, the *merged* compound itself, most probably, is not.

This poses a problem for most statistical parsers. In our work, we focus on recent neural dependency parsers which, instead of using hand-crafted feature templates, directly learn the features from the training data (Chen and Manning, 2014; Zhang et al., 2017). These parsers usually introduce an `UNKNOWN` token for out-of-vocabulary words. A well-known technique for computing the embeddings of the `UNKNOWN` token is to discard infrequent words below a certain *threshold* and also treat them as *unknown*.

To illustrate the differential effect of this practice for languages that write compounds with word-internal spaces versus languages that use run-together compounds, let us take a look at the English Penn Treebank (PTB) (Marcus et al., 1993) and the German data set from the SPMRL 2014 shared task (Seddah et al., 2014), and compare the ratio of sparse or unknown words in the test sets for both treebanks with regard to different frequency thresholds. Table 4.1 shows that the ratio of words to be declared *unknown* is more than twice as high in the German data, due to a high amount of unknown common nouns. At a cutoff threshold of 5, the most frequent POS tags for unknown words in the German data are common nouns (47.4%) and proper nouns (17.3%). In the English data, however, proper names are the most frequent source for `UNKNOWN`s (35.4%) and common nouns only amount to 24.1%. One of the main reasons for this difference between the two Germanic languages is the high productivity of German compounds. We thus hypothesize that the high ratio of compounds will have a major impact on parsing German, which we address with our new *compound* embeddings.

4.2 Character vs. Compound Embeddings

In a neural parsing system, each word is represented by a vector stored in a lookup table. One way to reduce the negative effect of unknown words in the vocabulary and also, if only indirectly, provide a treatment for compound words, is to replace the word lookup table with **character-based embeddings** (Ling et al., 2015b). In this approach, each word is treated as a sequence of characters, and the representation for each word is constructed from the representations for its characters, using a bidirectional Long Short-Term Memory Network (LSTM) (Hochreiter and Schmidhuber, 1997). Given a word w as a sequence of characters $(c_1, c_2, ..., c_m)$ and $\mathbf{e}_c$ as the vector representation of character c, we can compute the representation $\mathbf{e}_w^{\text{char}}$ of word w as follows:

$$\mathbf{s}_t^F = \text{LSTM}_F(\mathbf{e}_{c_t}, \mathbf{s}_{t-1}^F) \tag{4.1}$$

$$\mathbf{s}_t^B = \text{LSTM}_B(\mathbf{e}_{c_t}, \mathbf{s}_{t+1}^B) \tag{4.2}$$

$$\mathbf{e}_w^{\text{char}} = \mathbf{D}^F\mathbf{s}_m^F + \mathbf{D}^B\mathbf{s}_0^B + \mathbf{b} \tag{4.3}$$

where $\mathbf{s}_t^F$ and $\mathbf{s}_t^B$ are the hidden states of the forward and backward LSTMs at time t; $\mathbf{D}^F$ and $\mathbf{D}^B$ are the weight matrices, and $\mathbf{b}$ is the bias vector.

We now outline our compositional model for **compound embeddings**. We assume that most compounds have a transparent meaning that can be inferred from the meaning of their components and hypothesize that providing the parser with subword embeddings that combine the representations of the individual components will help the model to handle unseen compounds. To that end, we first split each compound into lexemes and then combine the sequence of lexemes as we did for the character-based embeddings, using a bidirectional LSTM.

For compound splitting, we use the IMS splitter (Weller and Heid, 2012) which adopts a frequency-based approach with additional linguistic features. The input

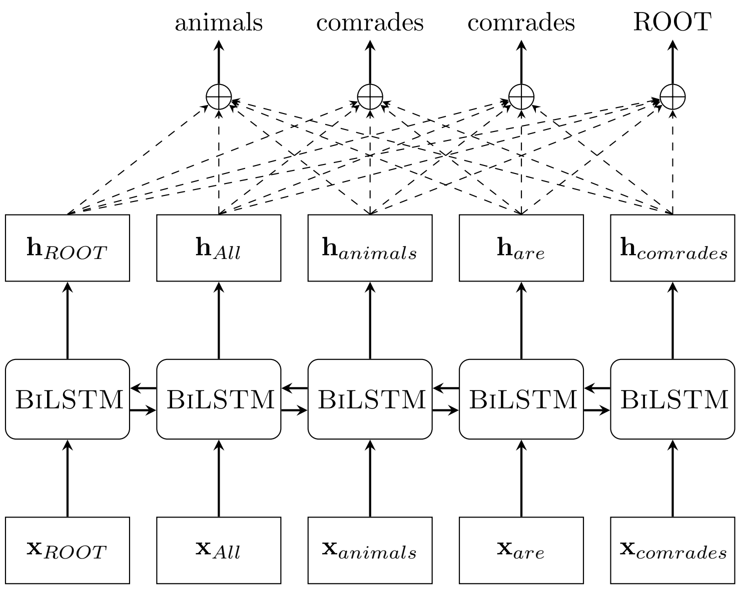


Figure 4.1: The parsing as head selection model

information for the splitter (frequencies, POS, and lemmas) was extracted from SdeWac (Faaß and Eckart, 2013), a cleaned-up version of the deWac corpus (Baroni et al., 2009) with automatic POS tags and lemmas.

4.3 Experiments

4.3.1 Parsing Model

Our parsing model is an extension of the *head-selection* parser of Zhang et al. (2017) (figure 4.1). Given a sentence $S = (w_0, w_1, ..., w_N)$ and $\mathbf{x}_t$ as the *input representation* of the word w_t, the model uses a bidirectional LSTM to learn a *feature vector* for each word in S:

$$\mathbf{h}_t^F = \text{LSTM}_F(\mathbf{x}_t, \mathbf{h}_{t-1}^F) \tag{4.4}$$

$$\mathbf{h}_t^B = \text{LSTM}_B(\mathbf{x}_t, \mathbf{h}_{t+1}^B) \tag{4.5}$$

$$\mathbf{h}_t = [\mathbf{h}_t^F; \mathbf{h}_t^B] \tag{4.6}$$

The feature vector $\mathbf{h}_t$ of the word w_t is the concatenation of the hidden states from the forward and backward passes of the bidirectional LSTM. An artificial root node w_0 token is appended at the beginning of each sentence.

Unlabeled parsing is modeled as choosing the most probable head for each word in a sentence. In the sentence $S = (w_0, w_1, ..., w_N)$, the probability of the word w_j

being the head of the word w_i is calculated as:

$$p_{\text{head}}(w_j \mid w_i, S) = \frac{\exp(f(w_j, w_i))}{\sum_{k=0}^{N} \exp(f(w_k, w_i))} \tag{4.7}$$

f is a neural network that predicts the score of an edge (w_j, w_i) given the feature vectors $\mathbf{h}_i$ and $\mathbf{h}_j$ as follows:

$$f(w_j, w_i) = \mathbf{v}^\top \cdot \tanh(\mathbf{U} \cdot \mathbf{h}_j + \mathbf{W} \cdot \mathbf{h}_i) \tag{4.8}$$

where $\mathbf{U}$ and $\mathbf{W}$ are weight matrices, $\mathbf{v}$ is a weight vector.

In a similar fashion, the probability of the edge (w_j, w_i) being assigned the grammatical function l is computed as:

$$p_{\text{label}}(l \mid w_j, w_i, S) = \frac{\exp(g(w_j, l, w_i))}{\sum_{l' \in \mathcal{L}} \exp(g(w_j, l', w_i))} \tag{4.9}$$

where $\mathcal{L}$ is the set of all labels. g is an additional neural network used to assign the grammatical function label to each edge in the unlabeled tree. The input to that network is the concatenation of the input representations and the learned feature vectors of the head j and the dependent i:

$$g(w_i, l, w_j) = \text{MLP}([\mathbf{x}_i; \mathbf{x}_j; \mathbf{h}_i; \mathbf{h}_j])[l] \tag{4.10}$$

Note that in our implementation we use a single hidden-layer rectifier network instead of the two-layer rectifier network in Zhang et al. (2017) since we achieve better results with only one hidden layer.

4.3.2 Input Representations

To assess the effect of different compound handling techniques on parsing performance, we systematically vary the input information for the parser, as described below:

Word Embeddings (+word) Each word w in the lexicon is represented as a vector e_w in the lookup table. We do not use any pre-trained embeddings; all embeddings are initialized randomly.

POS Embeddings (+pos) If word w has POS tag p, we add an embedding e_p for tag p to the input information.

Character-Based Embeddings (+char) In addition to the word embeddings e_w from the lookup table, we also use the character-based embeddings $\mathrm{e}_w^{\text{char}}$ of word w (equation 4.3).

Module	Hyperparameter	Value
Word emb.	size	300
POS emb.	size	40
Character-based emb.	character embedding size	50
	hidden size	100
Compound emb.	lexeme embedding size	50
	hidden size	100
BiLSTM	hidden size	300
Regularization	L2	1e-3
	input dropout rate	0.05
	dropout rate	0.5
	max-norm	5.0
Optimization	optimizer	Adam
	learning rate	0.001
	1st momentum	0.9
	2nd momentum	0.999
	no. epochs	15
Others	word cutoff threshold	5

Table 4.2: Hyperparameters used in all experiments

Compound Embeddings (+comp) The compound embedding $\mathbf{e}_w^{\text{comp}}$ of word w is calculated based on the lexeme embeddings (see section 4.2).

When combining different types of information in the input, we use the *concatenation* of each embedding type.

4.3.3 Training

We train our own implementation of the parser, following Zhang et al. (2017). For optimization, we used Adam (Kingma and Ba, 2015) with default parameters. All models were trained in 15 epochs and the training process was regularized using common techniques like L2 regularization, max-norm, and dropout (Srivastava et al., 2014). We chose all hyperparameters for our experiments manually, following suggestions by Zhang et al. (2017) (see table 4.2).

We report parsing performance (UAS and LAS) *with punctuation* on the German

		Input	UAS	LAS
+pos	**b1**	`+word,pos`	90.50	88.06
	b2	head:`+word,pos`	90.46	88.13
		`+comp,pos`	90.23	87.93
		`+comp,word,pos`	90.39	88.10
		`+char,pos`	90.53	88.49
		`+word,char,pos`	**90.69**	**88.56**
-pos	**b1**	`+word`	86.27	83.08
	b2	head:`+word`	87.00	83.97
		`+word,comp`	88.29	85.42
		`+word,char`	**90.42**	**88.20**

Table 4.3: Parsing results for different input combinations

data set from the SPMRL 2014 shared task. The training set contains 40,472 sentences and the development and test sets both include 5,000 sentences.

The compound splitting (section 4.2) affected about one-third of the lexemes in the lexicon, all of them nouns. Of all the unknown words in the test set (64.12% at a cutoff threshold of 5, see table 4.1), 24.92% now consist of known lexemes, 73.79% have only one unknown lexeme, and only 1.29% have more than one unknown component.

We compare our results against two baselines, (b1) using the original words for parsing and (b2) a greedy baseline *head*, where we discard all compound components except the rightmost one, since in most cases, the rightmost lexeme is the head of the compound. Baseline (b2) reduces the number of unknown words in the data by 10%.

4.3.4 Results

Table 4.3 (+pos) shows results for different combinations of input information. The `+word,pos` setting (baseline b1) is the one implemented in the original parser. The results show that our special treatment of compounds does not have the desired effect. In both settings, using only the head words (b2) and using compound embeddings, we see only minor changes in parsing accuracy and when replacing words with compound embeddings, the results actually decrease. This strongly suggests that the parsing model is often able to make the right decision without actually knowing the word.

Adding the character embeddings improves the LAS by 0.5%, but does not have

Label	Frequency	`-char`		`+char`	
		P	R	P	R
SB	6,638	**90.7**	91.2	90.6	**92.2**
OA	3,184	82.3	85.7	**83.3**	**87.5**
DA	568	73.2	55.3	**78.9**	**63.9**
AG	2,241	91.3	91.5	**94.2**	**93.9**
OG	21	**100.0**	**4.8**	N/A	0.0
PD	1,045	82.5	74.3	**84.8**	**80.8**

Table 4.4: Precision (P) and recall (R) for core grammatical functions with/without character-based embeddings. SB: subject, OA: accusative object, DA: dative object, AG: genitive attribute, OG: genitive object, PD: predicate.

a significant effect on the UAS. Since German is a richly inflected, semi-free word order language, this suggests that the character-based embeddings have learned *morpho-syntactic* information from the surface of the words which helps assign the correct grammatical function for each head-dependent pair. Table 4.4 confirms this by showing the improvements we get for the core arguments when adding character-based embeddings.

The effect of POS tags In the next set of experiments, we exclude the POS tag information to isolate the effect of the different techniques for handling compounds. Table 4.3 (-pos) shows that without POS information, we now see a significant effect. The greedy baseline that keeps only the head word for each compound increases UAS and LAS by 0.7% and 0.9% respectively, and our compound embeddings now improve both scores by more than 2%. Using character-based embeddings in combination with word embeddings, however, yields comparable results to the `+word,pos` system. We take that as evidence that the character-based embeddings implicitly learn morpho-syntactic information that is complementary to the information included in the word embeddings. Our results are in line with previous results from the literature, claiming that character-based embeddings are able to capture morphological information (Ling et al., 2015b; Cao and Rei, 2016; Kim et al., 2016).

Our results also corroborate findings by Köhn (2016) who evaluates different types of *word* embeddings in a syntax-based classification task, reporting that the embeddings yielded improvements *only* when no POS information was given.

Model	Word	Compound	Character
Perplexity	36.954	35.987	**32.273**

Table 4.5: Perplexity for different language models on German texts from Wikipedia.

4.3.5 Language Modeling

To validate our results in a different setting, we also test the compound embeddings in a language modeling task. Language models are an important ingredient in many NLP applications, e.g. in speech recognition and machine translation, and they require both syntactic and semantic information.

In our experiment, we use the framework[1] and setup described in (Vania and Lopez, 2017) to build a language model for German texts. The framework includes implementations for word and subword-based (morpheme, character or character n-gram) embeddings and uses either bidirectional LSTMs or addition as the combination function of subwords.

The German data sets are from the preprocessed Wikipedia data (Al-Rfou et al., 2013). Hyperlinks have been removed and the input texts have been lower-cased before learning the word- and compound-based embeddings. For the character-based embeddings, the upper-cased letters have been preserved. We split the data into training, development and test sets, with approximately 1.2M, 150K, and 150K tokens, respectively. For training and evaluation we closely follow Vania and Lopez (2017).

We report results for three language models. The *word* model and the *character* model (using a bidirectional LSTM as composition function)[2] are already implemented in the framework. For the *compound* embeddings, we first split the compounds in the data sets as described in section 4.2 and then combine them, again using a bidirectional LSTM as composition function.

The results are shown in table 4.5. Using compound-based embeddings yields better perplexity in comparison to the vanilla word model, but the compound model is still far behind the character-based embeddings which obtain the lowest perplexity. The results for the language model thus confirm the trend observed in the parsing experiments.

[1] `https://github.com/claravania/subword-lstm-lm`

[2] These are the same character-based embeddings that we used in the parsing experiment (sections 4.3.3, 4.3.4).

4.3.6 Discussion

In both tasks, parsing and language modeling, the character-based embeddings clearly outperformed the compound-based embeddings. This suggests that the character-based embeddings are able to pick up structural information that is important for both tasks.

For parsing, the results for the compound-based embeddings were even below the ones for the word embeddings when including POS information in the input. This implies that the information needed for *parsing* unknown words is not so much information about the semantics of a word but, crucially, morpho-syntactic information. This was confirmed by the improved results for using character-based embeddings instead of the compound-based ones, where we were able to make up for the decrease in LAS that resulted from removing POS information from the input.

Our results are important, as they show that unknown words are not per se a problem for parsing, as long as we are able to learn something about their morpho-syntactic properties.

4.4 Summary

In the chapter, we introduced a new type of subword embedding, the *compound* embedding, to deal with the challenge at the (sub)word and morphological level posed by unknown words when parsing MRLs. The new embeddings are designed to provide more information about unknown compounds which constitute a major part of OOV words in German.

We evaluated the embeddings in dependency parsing and showed that although the compound-based embeddings outperformed word embeddings when *no* POS information was available, the character-based model showed a performance superior to the one for word and compound embeddings. For language modeling, where not only syntactic but also semantic information is important, the results follow the same trend. The results indicate that at the (sub)word and morphological level, it is not the missing information about the semantics of the unknown words that causes problems for parsing German, but the lack of morpho-syntactic information for unknown words.

This leaves us with two avenues for future work. To provide improved handling of OOV words for parsing, we need to optimize subword embeddings to represent morpho-syntactic information for unknown words. In addition, we would like to test the compound embeddings in a purely *semantic* task where we can explore their full potential.

CHAPTER 5

Syntactic Level: Grammatical Function Labeling

In this chapter, we look into the next structural level of linguistics: the syntactic level. Specifically, we examine the effect of word order on a subtask of dependency parsing, which is *grammatical function labeling*.

For languages with a non-configurational word order and rich(er) morphology, such as German, grammatical function labels are essential for interpreting the meaning of a sentence. *Case syncretism* in the German case paradigm makes grammatical function labeling a challenging task. See (7) for an example where the nouns in the sentence are ambiguous between different cases, which makes it hard for a statistical parser to recover the correct reading.

(7)	Telefonkarten	bleibt	nichts	erspart.
	phone cards$_{Nom/Acc/Dat}$	remains	nothing$_{Nom/Acc}$	spared.

"Phone cards are subjected to all kinds of abuse."

We approach the problem of grammatical function labeling as a subtask of dependency parsing, where we first generate unlabeled trees and, in the second step, try to find the correct labels. This pipeline architecture gives us more flexibility, allowing us to use the labeler in combination with our parser, but also to apply it to the unlabeled output of other parsing systems without the need to change or re-training the parsers.

The approach also makes it straightforward to test different architectures for grammatical function labeling. We are especially interested in the influence of different input structures representing different (surface versus structural) orders of the input. In particular, we compare models where we present the unlabeled tree in linear order with a model where we encode the parser output as a tree. We show that all models are able to learn GFs with a similar overall LAS, but the model where the tree is encoded in a breadth-first order outperforms all other models on labeling core argument GFs.

5.1 Related Work

Grammatical function labeling is commonly integrated into syntactic parsing. Few studies have addressed the issue as a separate classification task. While most of them assign grammatical functions on top of constituency trees (Blaheta and Charniak, 2000; Jijkoun and de Rijke, 2004; Chrupała and van Genabith, 2006; Klenner, 2007; Seeker et al., 2010), less work has tried to predict grammatical function labels for unlabeled dependency trees. One of them is McDonald et al. (2006) who first generate the unlabeled trees using a graph-based parser, and then model the assignment of dependency labels as a sequence labeling task.

Another approach has been proposed by Zhang et al. (2017) who present a simple, yet efficient and accurate parsing model that generates unlabeled trees by identifying the most probable head for each token in the input. Then, in a post-processing step, they assign labels to each head-dependent pair, using a two-layer rectifier network.

Dependency Parsing as Head Selection Our labeling model is an extension of the parsing model of Zhang et al. (2017). We use our own implementation of the head-selection parser and focus on the grammatical function labeling part. The parser uses a bidirectional Long Short-Term Memory Network (LSTM) (Hochreiter and Schmidhuber, 1997) to extract a dense, positional representation a_i of the word w_i at position i in a sentence:

$$\mathbf{h}_t^F = \text{LSTM}_F(\mathbf{x}_t, \mathbf{h}_{t-1}^F) \tag{5.1}$$

$$\mathbf{h}_t^B = \text{LSTM}_B(\mathbf{x}_t, \mathbf{h}_{t+1}^B) \tag{5.2}$$

$$\mathbf{a}_t = [\mathbf{h}_t^F; \mathbf{h}_t^B] \tag{5.3}$$

$\mathbf{x}_i$ is the input at position i, which is the concatenation of the word embeddings and the tag embeddings of the word w_i. An artificial root token w_0 is added at the beginning of each sentence.

The unlabeled tree is then built by selecting the most probable head for each word. The score of the word w_j being the head of the word w_i is computed by a single hidden layer neural network on their hidden representations $\mathbf{a}_j$ and $\mathbf{a}_i$.

An additional classifier with two rectified hidden layers is used to predict dependency labels and is trained separately from the unlabeled parsing component, in

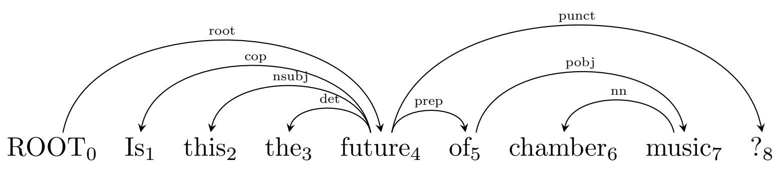


Figure 5.1: The dependency tree of the sentence *Is this the future of chamber music?*

a pipeline architecture. The classifier predictions are based on the representations of the head and the dependent, $\mathbf{b}_j$ and $\mathbf{b}_i$, which are the concatenation of the input and the bidirectional LSTM-based representations:

$$\mathbf{b}_i = [\mathbf{x}_i; \mathbf{a}_i] \tag{5.4}$$

Despite its simplicity and the lack of global optimization, Zhang et al. (2017) report competitive results for English, Czech, and German.

5.2 Labeling Dependencies with History

Although the labeling approach in Zhang et al. (2017) is simple and efficient, looking at heads and dependents only when assigning the labels comes with some disadvantages. First, some labels are easier to predict when we also take context into account, e.g. the parent and grandparent nodes or the siblings of the head or dependent.

Consider, for example, the following sentence: *Is this the future of chamber music?* and its syntactic structure (figure 5.1). If we only consider the nodes *this* and *future*, there is a chance that the edge between them is labeled as *det* (determiner). However, if we also look at the local context, we know that node *the* to the left of *future* is more likely to be the determiner, and thus *this* should be assigned a different label.

Second, when looking at the parser output, we notice some errors that are well-known from other local parsing models, such as the assignment of duplicate subjects for the same predicate. To address this issue, we propose an extended labeling model that incorporates a decision history. To that end, we design different LSTM architectures for the labeling task and compare their performance on German, Czech, and English.

Label prediction as a sequence labeling task Presenting the input to the labeler in sequential surface order does not seem very intuitive when we want to assign labels to a tree. This approach, however, was adapted by McDonald et al. (2006). In their work, they consider all dependents $x_{j1}, ..., x_{jM}$ of a head x_i and label those edges $(i, j1), ..., (i, jM)$ in a sequence.

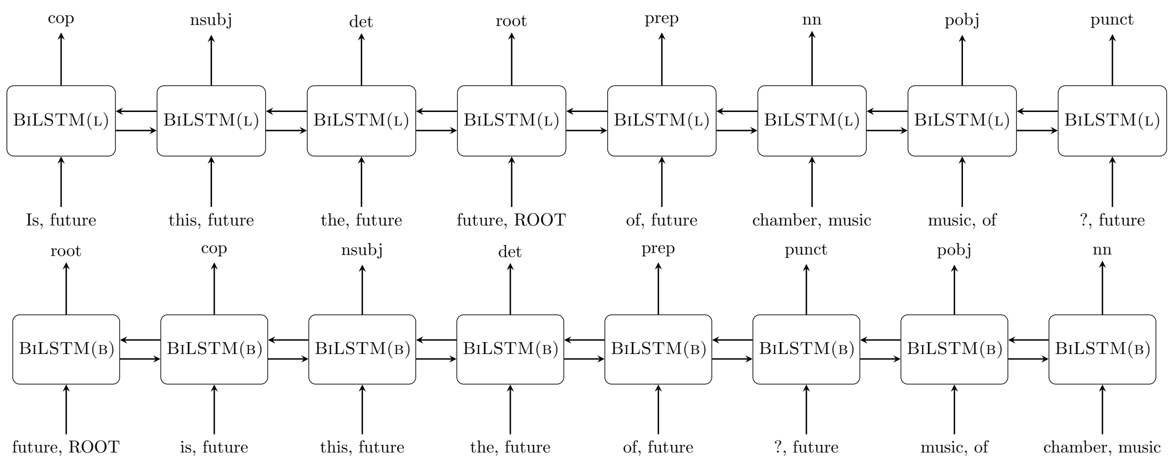


Figure 5.2: The processing order of the sentence in figure 5.1 (a) in the BiLSTM(L) model (top) and (b) in the BiLSTM(B) model (bottom).

We argue, however, that it is not enough to know the labels of the siblings, but that we also need to consider nodes at different levels in the tree. Therefore, when predicting the label for the current node, we consider all label decisions in the history and feed them to a bidirectional LSTM. Given a sequence of nodes $S = (w_1, ..., w_n)$ and their corresponding head $(h_1, ..., h_n)$, at each recurrent step, we input the learned representation of the head and the dependent:

$$\mathbf{h}_i^{F(\text{lbl})} = \text{LSTM}_{\text{lbl}}^{F}(\mathbf{b}_i, \mathbf{b}_{h_i}, \mathbf{h}_{i-1}^{F(\text{lbl})}) \tag{5.5}$$
$$\mathbf{h}_i^{B(\text{lbl})} = \text{LSTM}_{\text{lbl}}^{B}(\mathbf{b}_i, \mathbf{b}_{h_i}, \mathbf{h}_{i+1}^{B(\text{lbl})}) \tag{5.6}$$

After that, the concatenated hidden states $[\mathbf{h}_t^{F(lbl)}; \mathbf{h}_t^{B(lbl)}]$ are projected to a softmax layer to predict the label.

When presenting a tree as a sequence, we experiment with two different input orders:

- **BiLSTM(L)**: Tree nodes are ordered according to their surface order in the sentence (linear order; figure 5.2a).
- **BiLSTM(B)**: Tree nodes are ordered according to a breadth-first traversal (BFS) of the tree, starting from the root node (figure 5.2b). Here, siblings are closer to each other in the history.

Top-down tree LSTM Intuitively, it seems more natural to present the input as a tree structure when trying to predict the dependency labels. We do that by adopting the top-down tree LSTM model (Zhang et al., 2016) that processes nodes linked

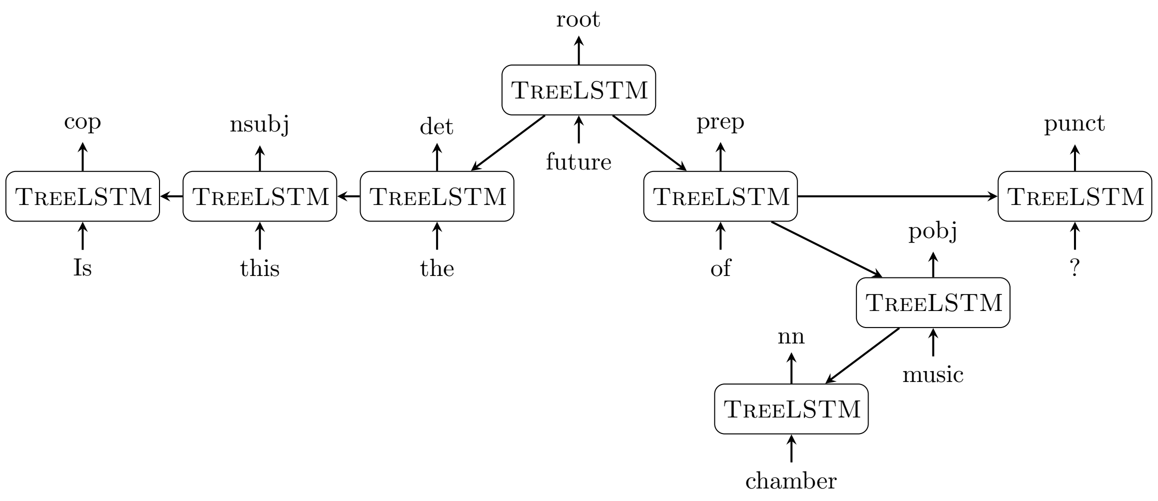


Figure 5.3: The processing order of the sentence in figure 5.1 in the TREELSTM model.

through dependency paths in a top-down manner. To make it comparable to the previous LSTM models, we only use *one LSTM* instead of four, and do not stack LSTMs. The hidden state is computed as follow:

$$\mathbf{h}_i^{(\text{lbl})} = \text{TREELSTM}(\mathbf{b}_i, \mathbf{h}_{i-1}) \tag{5.7}$$

After that, we proceed as we did for the **BILSTM** models (see above). Note that the processing order i is also the BFS order. We call this model **TREELSTM** (figure 5.3).

5.3 Experiments

Our interest is focused on German, but to put our work in context, we follow Zhang et al. (2017) and report results also for English, which has a configurational word order, and for Czech, which has a free word order, rich morphology, and less ambiguity in the case paradigm than German.

For English, we use the Penn Treebank (PTB) (Marcus et al., 1993) with standard train/dev/test splits. The POS tags are assigned using the Stanford POS tagger (Toutanova et al., 2003) with 10-way jackknifing, and constituency trees are converted to Stanford basic dependencies (De Marneffe et al., 2006). The German and Czech data come from the CoNLL-X Shared Task (Buchholz and Marsi, 2006) and our data split follows Zhang et al. (2017). As the CoNLL-X test sets are rather small (∼360 sentences), we also train and test on the much larger German SPMRL 2014 Shared Task data (Seddah et al., 2014) (5,000 test sentences). For the SPMRL data, we use the predicted POS tags provided by the shared task organizers.

5.3.1 Setup

We test different labeling models on top of the unlabeled trees produced by our re-implementation of the *parsing as head selection* model (section 5.1).

We first train the unlabeled parsing models for the three languages. Unless stated otherwise, all parameters are set according to Zhang et al. (2017), and tag embedding size was set to 40 for all languages. Please note that we do *not* use pre-trained embeddings in our experiments.

In the next step, we train four different labeling models: the labeler of Zhang et al. (2017) that uses a rectifier neural network with two hidden layers (**baseline**), two bidirectional LSTM models (**BiLSTM(l)** and **BiLSTM(b)**), and one tree LSTM model (**TreeLSTM**) (section 5.2).

The hidden layer dimension in all LSTM models was set to 200. The models were trained for 10 epochs and were optimized using Adam (Kingma and Ba, 2015) with default parameters (initial learning rate 0.001, 1st momentum coefficient 0.9, 2nd momentum coefficient 0.999). We used L2 regularization with a coefficient of 10^{-3} and max-norm regularization with an upper bound of 5.0. The dropout (Srivastava et al., 2014) rate was set to 0.05 for the input connections, and 0.5 for the rest.

5.3.2 Results

Table 5.1 shows the unlabeled attachment score (UAS) for the unlabeled trees and the labeled attachment scores (LAS) for the different labelers (excluding punctuation). All history-based labeling models perform significantly better than the local baseline model,[1] but for English, the improvements are smaller (0.3%) than for the non-configurational languages ($\sim$0.7%).

While we tried to re-implement the model of Zhang et al. (2017) following the details in the paper, our re-implemented model yields higher scores for German, compared to the results in the paper. The scores for English are slightly lower since, in contrast to Zhang et al. (2017), we do not use pre-trained embeddings. When using our history-based labelers, we get similar results for English (91.9%) and higher results for both Czech (84.1% vs. 81.7%) and German (91.0% vs. 89.6%) on the same data *without* using pre-trained embeddings or post-processing.

On the SPMRL 2014 shared task data, our results are only 0.3% lower than the ones of the winning system (Björkelund et al., 2014) *without* reranker (*blended*).[2] To further

[1]For significance testing, we use Bikel's Randomized Parsing Evaluation Comparator (`http://www.cis.upenn.edu/~dbikel/software.html`).

[2]The shared task winner is a complex ensemble system that generates a tree by *blending* the output of three parsers (Mate, Turbo, BestFirst; see Björkelund et al. (2014)).

Model	**en**	**cs**	**de**$_{\text{CoNLL}}$	**de**$_{\text{SPMRL}}$
UAS	93.35	89.70	93.09	91.29
Baseline	91.58	83.42	90.22	88.15
BiLSTM(l)	**91.92***	**84.08***	90.87*	88.73*
BiLSTM(b)	91.91*	83.80	**90.97***	**88.74***
TreeLSTM	**91.92***	83.82	90.89*	**88.74***
DenSe	91.90	81.72	89.60	-

Table 5.1: Results for different labelers applied to the unlabeled parser output. The first row reports UAS for the input to the labelers. The last row (DenSe) shows the results from Zhang et al. (2017). (*) indicates that the difference between the model and the baseline is statistically significant ($p < .001$).

illustrate the effectiveness of our models, we also ran our labeler on the *unlabeled output* of the SPMRL 2014 winning system and on *unlabeled gold* trees. On the output of the *blended* system LAS slightly improves from 88.62% to 88.76% (TreeLSTM).[3] When applied to *unlabeled gold* trees, the distance between our models and the baseline becomes larger and the best of our history-based models (BiLSTM(b), 97.38%) outperforms the original labeler of Zhang et al. (2017) (96.15%) by more than 1%.

We would like to emphasize that our history-based LSTM labeler is practically simple and computationally inexpensive (as compared to global training or inference), so our model manages to preserve simplicity while significantly improving labeling performance.

5.3.3 Discussion

Most strikingly, all three models seem to perform roughly the same, and the TreeLSTM model fails to outperform the other two models. However, in comparison to the BiLSTM models, the TreeLSTM model has a smaller number of parameters, and the history only flows in one direction. The tree model also has a shorter history chain since nodes are linked by paths from the root (figure 5.3), which might explain why it does not yield better results than the linear LSTM models.

The overall results suggest that the order in which the nodes are presented in the history does not have any impact on the labeling results. However, when looking at results for individual core argument functions (subject, direct object, etc.), a more

[3]Following Björkelund et al. (2014), here we include punctuation in the evaluation.

de$_{\text{SPMRL}}$	**SB**	**OA**	**DA**	**PD**
	# 6,638	# 3,184	# 568	# 1,045
baseline	90.3	83.6	64.7	77.1
BiLSTM(l)	91.4	85.3	67.7	80.0
BiLSTM(b)	**91.9**	**85.4**	**69.3**	**80.5**
TreeLSTM	91.2	85.1	68.6	79.8
de$_{\text{SPMRL}}$	**AG**	**PG**	**OC**	**OG**
	# 2,241	# 388	# 3,652	# 21
baseline	91.3	80.0	90.1	0
BiLSTM(l)	91.3	81.6	90.5	16.0
BiLSTM(b)	**91.5**	**82.4**	**90.7**	**37.0**
TreeLSTM	91.4	81.4	90.2	27.6

Table 5.2: LAS for core argument functions (German SPMRL data), and frequency (#) of grammatical functions in the test set (SB: subject, OA: accusative object, DA: dative object, PD: predicate, AG: genitive attribute, PG: phrasal genitive, OC: clausal object, OG: genitive object).

pronounced pattern emerges (table 5.2).[4] Here we see the benefit of encoding the siblings close to each other in the history: For all core argument functions, the BiLSTM(b) model outperforms the other models.

To find out why the history-based models work better for Czech and German than for English, we compared the average dependency length as well as the variability in head direction (how often e.g. the head of a subject is positioned to the left, in relation to the total number of subjects). Table 5.3 suggests that the success of the history-based models is not due to better handling of long dependencies but that they are better in dealing with the uncertainty in head direction (also see Gulordava and Merlo (2016)).

5.4 Summary

We have shown that grammatical function labeling, which is of crucial importance for languages like German, can be improved by combining LSTM models with a decision history. All our models outperform the original labeler of Zhang et al. (2017)

[4]We evaluate GFs on the German SPMRL data which are sufficiently large with 5,000 test sentences. The CoNLL data sets, in comparison, only include ∼360 test sentences.

	GF	**en**	**cs**	**de**$_{\text{SPMRL}}$
	sb	3.1	3.4	3.9
dep-length	dobj	2.5	*2.4	4.2
	iobj	1.7	-	4.7
	sb	4.6	32.5	34.2
left-head ratio	dobj	97.4	*77.5	37.2
	iobj	100.0	-	27.5

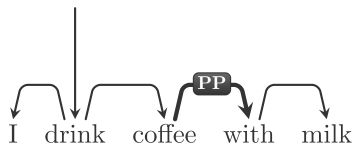
Table 5.3: Average dependency length and ratio of left arcs vs. all (left + right) arc dependencies for arguments. (* in the Czech data, *dobj* subsumes all types of objects, not only direct objects)

and give results in the same range as the best system from the SPMRL 2014 Shared Task (without the reranker), but with a much simpler model. Our results show that the history is especially important for languages that show more word order variation. Here, presenting the input in a structured BFS order not only significantly outperforms the baseline, but also yields improvements over the other LSTM models on core grammatical functions.

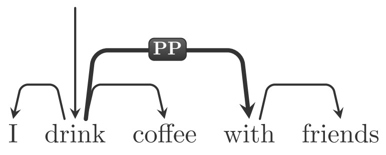
CHAPTER 6

Semantic Level: PP Attachment Disambiguation

Prepositional phrase (PP) attachment disambiguation, the task of identifying the correct attachment site for each preposition in the syntax tree, has often been described as the canonical case of structural ambiguity in NLP, with crucial impact on semantic interpretation. Consider the example in figure 6.1a. In this sentence, the PP *with milk* can either be attached to the verb *drink* or the noun *coffee*. The figure shows the correct head of the PP. Even though the PP attachment problem has been studied since the nineties (Hindle and Rooth, 1993; Brill and Resnik, 1994; Ratnaparkhi et al., 1994), it is still one of the hardest problems for syntactic parsing. For constituency parsing of English, Kummerfeld et al. (2012) showed that PP attachment errors are the largest error category across all parsers included in their evaluation. This also holds for dependency parsing. In experiments with the biaffine parser of Dozat and Manning (2017) on German, we found that 21.8% of the unlabeled attachment errors are due to incorrect PP attachments. What makes PP attachment such a challenging task is that *morpho-syntactic* information is often insufficient to resolve the ambiguity, in particular for (semi-)free word order languages, and additional *semantic* information or even *world knowledge* is needed. Figure 6.1 shows two sentences with only one word difference, but this word determines the PP attachment site in the sentence. To identify the correct attachment site for each preposition, the parser either needs to see these head-PP pairs during training (i.e., *lexical information*) or has to be able to deduce the correct attachment based on seen examples (i.e., *semantic information*). We can improve lexical coverage by adding more training data, however, treebanking is extremely time-consuming and requires linguistic expertise. In addition, most PPs are adjuncts, which means that they can choose their heads more freely (in comparison to arguments) and it is thus harder to achieve sufficient coverage. Neural network parsers, which take advantage of external (contextualized) word embeddings, can



(a)



(b)

Figure 6.1: Examples of PP attachment. Two sentences with similar words have two different PP attachment structures.

overcome some obstacles in lexical coverage. A different approach tries to improve PP attachment accuracy by modeling the problem as a separate task. This has the advantage that we do not need fully annotated parse trees as input, and that such a setup makes it easy to integrate a wide range of heterogeneous features. Many studies have tried to solve this task, however, only a few have shown that their system is able to improve the output of a strong syntactic parser (see section 6.1).

In this chapter, we attempt to improve dependency parsing at the **semantic** level via PP attachment disambiguation. We present a new PP attachment disambiguation system, based on biaffine attention and contextual word embeddings. While obtaining substantial improvements over previous work, we show that modeling all head-dependent pairs jointly (as done in full parsing) allows the system to make more effective use of the training data and is thus superior to modeling PP attachment as a separate task.

We first review statistical methods for PP attachment (section 6.1), focusing on German (section 6.2). After outlining some shortcomings of recent work, we reproduce a state-of-the-art PP attachment disambiguation system and evaluate it in a realistic scenario, comparing its performance to that of a strong neural parser (section 6.3). In section 6.4 we propose a new approach based on contextualized word embeddings that overcomes limitations of previous work, and we summarize our results in section 6.5.

6.1 Related Work

More than two decades have passed since Hindle and Rooth (1993) presented one of the first statistical models to resolve the ambiguity in PP attachment. Since then, many approaches have been proposed to disambiguate PP attachment, from lexical association and traditional statistical models to word embeddings and neural networks. In this section, we give an overview of previous works on PP attachment disambiguation, from the way the problem is framed to the features and models used to solve the problem.

6.1.1 Problem Formulation

So far, all previous studies on PP attachment only consider nouns and verbs as possible heads for a PP. Although a PP can be attached to other word categories and appear in other constructions, these cases are less frequent and are usually ignored. Early work on PP attachment disambiguation (Hindle and Rooth, 1993; Brill and Resnik, 1994; Ratnaparkhi et al., 1994) traditionally formulated the task as a binary choice between a given *verbal* and a *nominal* head candidate while ignoring other parts of speech as possible attachment sites, as well as other potential verbs or nouns in the same sentence that might also be attachment candidates. For example, Brill and Resnik (1994) and Ratnaparkhi et al. (1994) formally define the task input as a quadruple (v, n_1, p, n_2) where v and n_1 are a verbal and a nominal head candidate, p is the preposition and n_2 is the head of the object of p. There are also works that try to disambiguate without considering the object of the preposition (Hindle and Rooth, 1993; Ratnaparkhi, 1998).

The mechanism that extracts the two candidate attachment sites is called *oracle*. In most works, the oracle simply extracts these quadruples from gold information. This oracle-based approach is argued to be unrealistic since such oracles do not exist in real applications. Atterer and Schütze (2007) compare a full parsing setup with PP attachment systems, and see no statistically significant difference between the performance of the parser and the PP attachment systems for ambiguous cases detected by the parser. Recent works, therefore, use an improved setup where they look at multiple attachment sites. Belinkov et al. (2014) extract all nouns and verbs in a 10-word window surrounding the preposition as candidates, which covers 99% of the PP attachments in the Penn Treebank (PTB). de Kok et al. (2017a) use the topological field distribution of prepositions and their heads to construct a PP attachment data set with multiple candidate heads (see more details about this data set in section 6.2).

Using only the head words without the full sentence as input does not provide enough information for disambiguation, even for human annotators. Ratnaparkhi et al. (1994) show that the performance of human experts drops from 93.2% to 88.2% on 300 sentences if they are restricted to use only head words. Olteanu and Moldovan (2005) extract the full noun phrases preceding the verb np_1 and the prepositional phrase np_2 and include features like the path between v and np_1, the label of the parent of np_1... Belinkov et al. (2014) experiment with a model that looks at the word following the candidate head, but its performance is inferior to a model that only considers head words.

6.1.2 Features

A diverse set of features has been proposed in the literature.

Lexical association Lexical associations have been widely used to resolve ambiguity in NLP. They utilize knowledge from an external, large corpus to compensate for limited training data in supervised approaches. Hindle and Rooth (1993) parse a 13 million word sample of Associated Press news stories from 1989 and extract more than 200,000 (v, n_1, p) triples. An unsupervised, iterative method is used to assign the PP attachment to either the noun or the verb by first deciding on the unambiguous cases, then assigning the rest by considering the log-likelihood:

$$LA(v, n_1, p) = \log_2 \frac{P(\text{verb attach } p \mid v, n_1)}{P(\text{noun attach } p \mid v, n_1)} \tag{6.1}$$

where:

$$P(\text{verb attach } p \mid v, n_1) \approx P(p \mid v) \cdot P(\text{NULL} \mid n_1) \tag{6.2}$$

$$P(\text{noun attach } p \mid v, n_1) \approx P(p \mid n_1) \tag{6.3}$$

Other work such as Ratnaparkhi (1998) only extracts data from *unambiguous* cases from the 1988 Wall Street Journal corpus where there is only one attachment site possible based on the predictions of a POS tagger and a chunker. It is still unclear whether using only unambiguous data is better than using additional, possibly noisy attachment information. Pantel and Lin (2000) use a parser to process a 125 million word newspaper and add all possible attachment sites of a preposition p to construct an ambiguous data set. They show that ambiguous data are beneficial if they are used alongside unambiguous data (constructed similar to Ratnaparkhi (1998)). Later, de Kok et al. (2017b) achieve their best results on a German PP data set by using point-wise mutual information computed on unambiguous data.

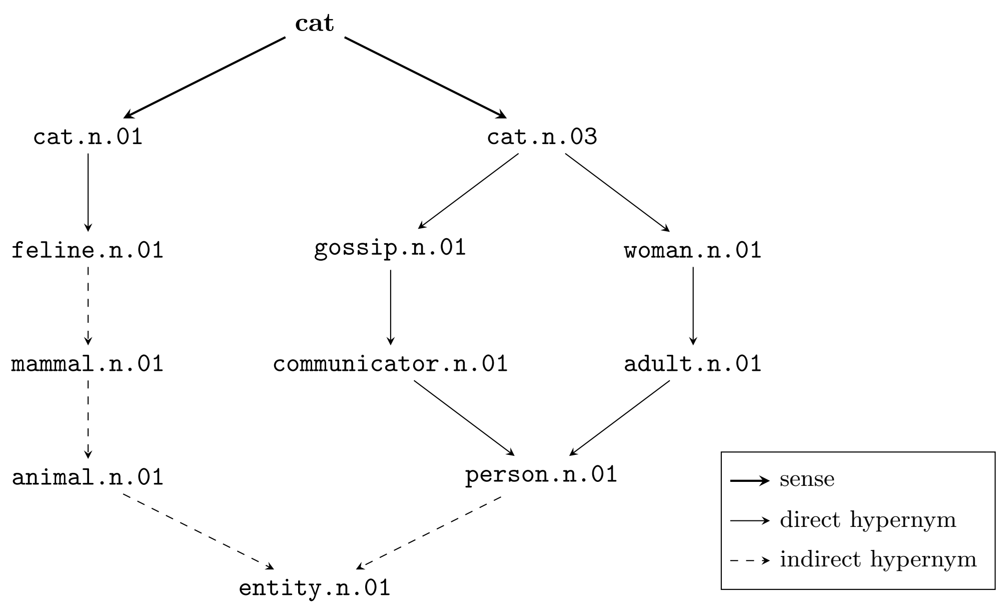


Figure 6.2: The hierarchy of the word *cat* with two senses in WordNet 3.1. Other senses and some intermediate hypernyms were removed for simplicity.

Besides computing the score using tuples extracted from a corpus, lexical association can be estimated via the co-occurrence of those words in a very large corpus. The World Wide Web (WWW) is often chosen as a corpus and can be queried with a search engine (Volk, 2001; Olteanu and Moldovan, 2005). For instance, Volk (2001) compares the co-occurence score $cooc(n_1, p, n_2)$ and $cooc(v, p, n_2)$ to decide the attachment of p:

$$cooc(x, p, n_2) = \frac{f(x, p, n_2)}{f(x)} \tag{6.4}$$

where $f(\cdot)$ is the number of results returned by the search engine when querying "x p n_2", x is either v or n_1.

Semantic information Data sparsity is a well-known problem of PP attachment. Using a statistical n-gram model with back-off, Collins and Brooks (1995) achieve an accuracy of 84.1% on 3,097 test sentences. Among them, the accuracy of quadruple and triple matches is more than 90%, but they only account for less than 30% of the cases. *Semantic word class* is a way to overcome this problem. Words that are semantically related can be used to deduce the attachment site when they appear in similar contexts. For example, knowing that the head of *with* is *coffee* in context *drink coffee with milk*, one can reach the same conclusion for the tuple *drink coffee with sugar* because *milk* and *sugar* are both in the category of *food*. WordNet (Fellbaum, 1998) is a lexical resource for English, in which similar word senses are group into a unique concept called *synset* (synonym set). Synsets are connected by relations, notably hypernym/hyponym (or *is-a*) relations that form a hierarchy of concepts. Figure 6.2 illustrates the hierarchy of two senses of the word *cat*. Adding synsets as features besides words to a transformation-based error-driven learning system improves the accuracy from 80.1% to 81.1% on 500 test samples extracted from the PTB (Brill and Resnik, 1994). Belinkov et al. (2014) enrich word representations with a binary dimension indicating the top hypernym of the word.

In order to use synsets effectively, the correct sense of the word in a context must be determined. Stetina and Nagao (1997) perform word sense disambiguation (WSD) for PP quadruples based on the *semantic distance* calculated based on the WordNet hierarchy:

$$D = \frac{1}{2}\left(\frac{L_1}{D_1} + \frac{L_2}{D_2}\right) \tag{6.5}$$

where L_1 and L_2 are the lengths of the path from the two words and their lowest common ancestor, and D_1 and D_2 are their depth (distance to the root) in the hierarchy. For each ambiguous word in a quadruple, the most similar quadruple is selected (based on semantic distance) and the ambiguous word takes the nearest sense of the corresponding word in the nearest quadruple. Agirre et al. (2008) experiment with both synsets and supersenses (broad semantic categories of synsets) of WordNet as fine and coarse-grained semantic representation. They disambiguate word senses with gold standard senses, most frequent senses, and using an automatic sense reranker. The results show that automatic WSD even outperforms gold standard senses overall, in both parsing and PP attachment settings. Dasigi et al. (2017) combine the representation of all hypernyms of a word (direct and indirect) in a weighted sum to compute the word representation. The weights are *context-sensitive*, and are computed in a similar way to the attention mechanism (Bahdanau et al., 2015) (which returns higher weights for components that are more relevant to the current context).

In addition to WordNet, automatically predicted word class information based on mutual information clustering is also used as a feature for PP attachment (Ratnaparkhi et al., 1994).

A different way to tackle lexical sparsity is to extend the lexicon with *semantically similar words*. Pantel and Lin (2000) use a syntactic collocation database and a corpus-based thesaurus to construct a list of contextually similar words for v, n_1 and n_2. The attachment score is computed not only on the original words of a quadruple but also on the list of contextually similar words.

Semantic information of verbs has also been used as features in PP attachment disambiguation systems. Olteanu and Moldovan (2005) use the semantic frame of

v and the semantic and thematic roles of n_1 extracted from FrameNet (Baker et al., 1998) as features when experimenting on the same data. Belinkov et al. (2014) extend word embeddings with a binary feature for VerbNet (Schuler, 2005) to show whether the frame of the candidate head contains a preposition.

Syntactic information Syntactic features have been used to recover missing context information in the extracted system input. Examples are the distance between the candidate head and the preposition (Olteanu and Moldovan, 2005; Belinkov et al., 2014; de Kok et al., 2017b), verb subcategorization (Olteanu and Moldovan, 2005), and topological fields (de Kok et al., 2017b).

Pre-trained word embeddings Low dimensional, dense word embeddings that map similar words to similar real vectors have played a successful part in many NLP tasks, such as parsing (Socher et al., 2013; Chen and Manning, 2014). When trained on a large, external corpus, they provide an effective way to enrich lexical information and are frequently used as the input to PP disambiguation systems with a neural network architecture (Belinkov et al., 2014; Dasigi et al., 2017; de Kok et al., 2017b). In experiments with German data, de Kok et al. (2017b) shows that the external word embeddings increase the lexical coverage of the test set from 71.7% to 89.5%, an improvement over the baseline without embeddings of nearly 14%.

6.1.3 Models

A variety of statistical models have been employed in PP attachment systems. We summarize the models used in previous (selected) works in table 6.1. Besides classical models, neural networks have also been used as a classifier in PP attachment systems (Alegre et al., 1999; de Kok et al., 2017b). For instance, Belinkov et al. (2014) use recursive neural networks (Socher et al., 2010) to compute the phrase representation of a candidate head with respect to the preposition and its object. Their best model, Head-Prep-Child-Dist (HPCD), computes the phrase representation as follows:

$$\mathbf{p}_1 = g(\mathbf{W}[\mathbf{p};\mathbf{c}] + b) \tag{6.6}$$

$$\mathbf{p}_2 = g(\mathbf{W}^d[\mathbf{h};\mathbf{p}_1] + b^d) \tag{6.7}$$

where $\mathbf{h}$, $\mathbf{p}$, and $\mathbf{c}$ are the vector representation of the candidate head, the preposition, and the preposition object. $\mathbf{W}$ and $\mathbf{W}^d$ are weight matrices, b and b^d are biases, g is a non linear function. $\mathbf{W}^d$ and b^d depend on the distance d from the candidate head to the preposition. Dasigi et al. (2017) employ Long Short Term Memory networks (LSTMs) (Hochreiter and Schmidhuber, 1997) to encode the context of a candidate

System	Model	External Resources
Hindle and Rooth, 1993	MLE	
Brill and Resnik, 1994	transformation based error driven learning	WordNet
Ratnaparkhi et al., 1994	MLE	
Collins and Brooks, 1995	MEE	
Stetina and Nagao, 1997	decision tree	WordNet
Ratnaparkhi, 1998	MLE	
Alegre et al., 1999	feed-forward NNs, ensemble	WordNet
Pantel and Lin, 2000	MLE	collocation db., thesaurus
Volk, 2001	MLE	WWW
Olteanu and Moldovan, 2005	SVM	FrameNet, WWW
Agirre et al., 2008	parsing	WordNet
Belinkov et al., 2014	recursive NNs	WordNet, VerbNet
Dasigi et al., 2017	recurrent NNs	WordNet
de Kok et al., 2017b	feed-forward NNs	

Table 6.1: A summary of approaches in PP attachment disambiguation

head for their context-sensitive sense representation (see section 6.1.2). However, the input sequence to the LSTMs is not the original sentence, but $t = (h_1, ..., h_K, p, d)$, where h_i is the i-th candidate head, p is the preposition and d is the object of the preposition.

6.1.4 Comparison with Syntactic Parsing

Early work focuses on evaluating PP attachment as an independent task rather than trying to compare with or integrate it with parsing. Exceptions are Foth and Menzel (2006) and Roh et al. (2011) who integrate lexical preferences in rule-based parsers. Disambiguation systems that rely on an *oracle* (the mechanism that extracts the two candidate attachment sites, based on gold standard data) have been criticized by Atterer and Schütze (2007). The authors argue that using gold information for candidate extraction is highly unrealistic as it will always include the correct solution in the candidate set, while in real applications the correct solution might not be available. Therefore, PP attachment disambiguation systems should be used to refine the preliminary syntactic analysis of a sentence provided by a parser, or to *reattach*

PP attachments. Their experiments with three PP reattachment systems show that none of the systems was able to obtain a significant improvement over the baseline parser. Agirre et al. (2008) later follow this evaluation setup and report a small but significant improvement in parsing accuracy using sense information.

Recently, PP attachment studies abandoned the binary setup based on gold-standard oracle due to its limitations. Recent works on English evaluate their performance on a data set with more than two attachment sites (Belinkov et al., 2014) and report small improvement over a baseline parser (Belinkov et al., 2014; Dasigi et al., 2017).

Up to now, the PP attachment problem has been studied extensively, with the latest models incorporating novel techniques like neural networks and word embeddings. Most proposed approaches revolve around improving lexical coverage, either by utilizing large, external corpora or by integrating semantic information. Despite the fact that some systems report better results than a baseline parser, research on PP attachment still falls behind those for state-of-the-art syntactic parsing. In recent work on PP attachment, Dasigi et al. (2017) employ a non-neural network parser with 94.17% accuracy on the Penn Treebank (using gold POS tags) as a baseline parser while the state-of-the-art (without contextualized embeddings) is 96.09% (Zhou and Zhao, 2019). Moreover, while most syntactic parsers perform well with predicted information like POS tags, PP attachment systems are usually evaluated in setups with gold information. As a final point, it is worth noting that restricting PP attachment sites to only nouns and verbs is very limited compared to full parsing.

6.2 PP Attachment in German

In this section, we take a closer look at PP attachment studies for German. Due to a freer word order, not all techniques in section 6.1.2 can be easily applied for German, for example, using a window to extract candidates as in Belinkov et al. (2014). Notable works for PP attachment in German include de Kok et al. (2017a) and de Kok et al. (2017b), where the former introduces a method to build a realistic PP attachment data set for German, and the latter reports strong disambiguation results on this data set. Details about the data set and the model are presented in the following sections.

6.2.1 Extracting a PP Attachment Data Set for German

De Kok et al. (2017a) create their new data set from the dependency version of the TüBa-D/Z treebank (Hinrichs et al., 2004), aiming at a more realistic setup

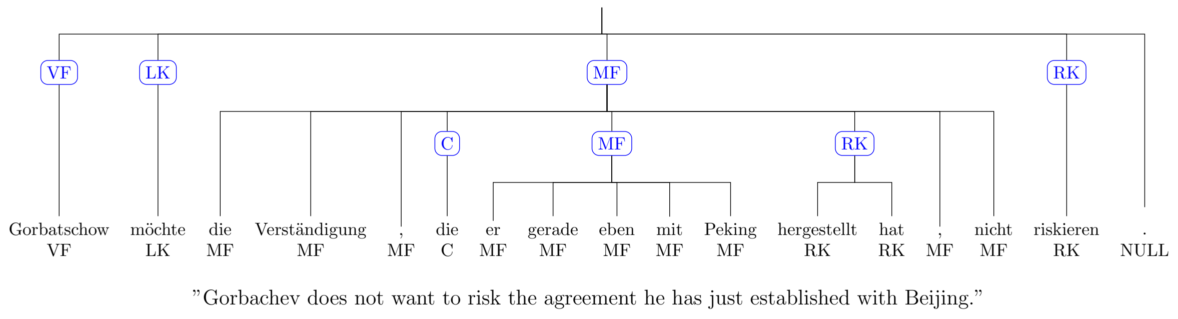


Figure 6.3: The topological field structure of a sentence from the TüBa-D/Z corpus and the corresponding projected topological field tags

Type	Topological fields						
VL:	KOORD			C	MF	RK	NF
V1:	KOORD	LV		LK	MF	RK	NF
V2:	KOORD	LV	VF	LK	MF	RK	NF

Table 6.2: The topological field model of German. C: complementizer, KOORD: coordination field (*Koordinationsfeld*), LV: topicalization (*Linkversetzung*).

where multiple potential attachment sites in the sentence are considered, rather than modeling PP attachment disambiguation as a binary classification problem. The authors argue that including *all* nouns and verbs in the sentence as attachment candidates is unnecessary, given that topological word order constraints for German make some positions very unlikely candidates. Instead, they propose to use the topological field model (Drach, 1937; Höhle, 1986), a grammar theory modeling German sentence structure, to extract only those candidates in the sentence that are probable attachment sites, based on the distribution of PPs and their heads across the topological fields annotated in the TüBa-D/Z.

Topological fields The topological field theory (Drach, 1937; Höhle, 1986) is a descriptive theory about the order of constituents in Germanic languages based on the distribution of verbs in a sentence. Although the word order in German is relatively free, it still follows the restrictions described by the topological field model. Namely, the verbal components of a German sentence form the sentence *brackets*, and the other topological fields are defined relative to these brackets. The sentence brackets include the *left bracket* (*linke Klammer*, LK) which is the finite verb and the *right bracket* (*rechte Klammer*, RK) which is the verb complex. The *initial field* (*Vorfeld*, VF) appears before the LK, the *final field* (*Nachfeld* NF) stands after the RK, and the *middle field* (*Mittefeld*, MF) is between the LK (or the complementizer) and the RK.

			Preposition field		
			VF	MF	NF
Head field	nominal	VF	41.16	0.24	0.57
		MF	1.73	33.47	6.15
		NF	0.00	0.05	35.74
	verbal	LK	55.24	22.19	18.17
		RK	1.87	44.05	39.37

Table 6.3: Distribution of prepositions and their heads in the TüBa-D/Z corpus (reproduced from de Kok et al., 2017a)

German clauses are classified into three types (Höhle, 1986): verb-last (VL), verb-first (V1), and verb-second (V2). VL clauses include subordinate clauses, V1 clauses contain imperatives and yes-no questions, and V2 clauses consist of declarative sentences and W-questions. The topological field structures of these clause types are listed in table 6.2. Figure 6.3 illustrates a topological field structure of a sentence in German.

The topological field distribution of the prepositions and their heads in the corpus (table 6.3) comprises many interesting properties:

- If the head of a preposition is a noun, it stays mainly in the same topological field as the preposition. Specifically, if a preposition is in the MF, its nominal head is rarely in the VF or NF. If the preposition is in the VF or the NF, the nominal attachment lies in the same field for most of the cases, but the preposition still can be attached to a noun in the MF.

- The reason that a preposition in the VF or NF can have a nominal head in the MF is because of topicalization. If the PP is not topicalized, the overall picture becomes clearer. Table 6.4 illustrates a typical case of non-topicalized PPs, and it can be seen that the nominal heads again are in the same topological field as the prepositions.

- The verbal attachment of a preposition is either in the LK or the RK, depending on where the main verb is.

- 97.5% of the nominal attachments in the MF are leftward.

- When the head of a preposition is a noun, in 12.41% of the case there is another noun between the head and the preposition.

			Preposition field	
			VF	NF
Head field	nominal	VF	98.34	0.07
		MF	0.01	0.47
		NF	0.00	94.85
	verbal	LK	1.65	1.74
		RK	0.00	2.88

Table 6.4: Distribution of prepositions and their heads in the TüBa-D/Z corpus when the preposition is in the VF or NF and is immediately preceded by a noun (reproduced from de Kok et al., 2017a)

The observations described above constitute the rules to extract the instances for the PP attachment data set. In the end, the data set contains 72,878 prepositions with at least two candidate heads per preposition. The head extraction rules on average reduce the number of candidates from 10.34 to 3.15. de Kok et al. (2017a) compare their approach to very early approaches on PP attachment where the nominal candidate head of a preposition is the immediately preceding noun, and the task is binary classification. By removing all the instances where either the preposition is not preceded by a noun or the head of the preposition is not the immediately preceding noun, about one-third of the data are incorrectly classified because of the naive assumption of closest attachment. The classification accuracy on the binary data set is 10% higher than that on the multiple candidate data set (using the same architecture trained on the same train/dev sizes), which confirms that considering multiple candidate heads is the more difficult and realistic setup for PP disambiguation.

Although not being emphasized, de Kok et al. (2017a) consider only nouns and verbs as possible candidate heads. In addition, the oracle they use to extract candidate heads relies on gold information, i.e., the gold POS, topological field tags, and parse trees.

6.2.2 PP Attachment Disambiguation for German

Model De Kok et al. (2017b) present a *neural scoring model* to estimate the probability of a candidate to be the correct attachment site. The input to the system is a triple <preposition, object of the preposition, candidate>. The candidate with the highest score given by the scoring model is returned as the correct head of the preposition.

The scoring model is a feed-forward neural network with one hidden layer. The hidden layer uses the ReLU activation function (Hahnloser et al., 2000), while the output layer is transformed with the logistic function to return probabilities between 0 and 1. The model is trained using the cross-entropy loss with Adagrad (Duchi et al., 2011). Dropout (Srivastava et al., 2014) and batch normalization (Ioffe and Szegedy, 2015) are applied as regularization.

Features The following features are used in the system:

- Basic features: including the word form and POS tag of the preposition, the object, and the candidate head; the logarithm of the *absolute distance* (the number of words) between the preposition and the candidate; the *relative distance* (the number of competing candidates) between the preposition and the candidate.
- Word and POS tag embeddings, both trained on a joint of two German corpora: TüBa-D/W (de Kok, 2014) and TüPP-D/Z (Müller, 2004) (800 million tokens in total) with a variation of `word2vec` (Mikolov et al., 2013b).
- Topological field tags: The topological field structure from the TüBa-D/Z treebank is linearized by projecting the nearest topological field as a tag (see figure 6.3). The topological field tags of the preposition, the object, and the candidate are represented by one-hot vectors.
- Auxiliary distributions: Pre-trained dependency parser is used to parse a large corpus of German newswire *taz* from 1986 to 2009 (28.8 million sentences, 393.7 million tokens). The triples of <preposition, object of the preposition, head of the preposition> are extracted from the parser's predictions, from both unambiguous and ambiguous cases. Unambiguous cases are those where there is only one possible attachment site for the PP (Ratnaparkhi, 1998). The triples are used to compute the bilexical interaction between three pairs: (candidate, preposition), (candidate, object), and (candidate, preposition+object) using the normalized point-wise mutual information (NPMI) (Bouma, 2009):

$$SI(x,y) = \ln \frac{p(x,y)}{p(x)p(y)} \bigg/ -\ln p(x,y) \tag{6.8}$$

  The last pair considers the preposition and its object combination as one token. In addition to the bilexical interaction, the trilexcial association scores between three elements are also included using the interaction information and the total correlation (Van de Cruys, 2011), which are the generalization of PMI for multivariate distributions. The interaction information of three variables is defined as:

$$SI_1(x,y,z) = \log \frac{p(x,y)p(y,z)p(z,x)}{p(x)p(y)p(z)p(x,y,z)} \tag{6.9}$$

de Kok et al., 2017b			Our reproduction	
Name	Model	Accuracy	Name	Accuracy
NN1	NN with one-hot vectors	68.2		
NN2	NN with embeddings	82.0		
NN3	NN2 + topological fields	83.8	PP-REP	84.1
NN4	NN3 + auxiliary all	86.5	PP-REP-AUX	**86.8**
NN5	NN3 + auxiliary unamb.	**86.7**		

Table 6.5: PP attachment disambiguation results for different settings in the TüBa-D/Z corpus (reproduced from de Kok et al., 2017b) and our reproduced results

The total correlation of three variables is defined as:

$$SI_2(x, y, z) = \log \frac{p(x, y, z)}{p(x)p(y)p(z)} \tag{6.10}$$

In total, the five association scores are included in the feature set.

Training & Evaluation After removing sentences used to train the parser for creating the auxiliary distributions, 43,845 instances[1] are left from the original data set of de Kok et al. (2017a). The remaining instances are split so that 80% are used for training and 20% are used for testing. Initially, a part of the training set is used as a development set for hyperparameter tuning. After that, a model is trained on the full training set and is evaluated on the original test set. The results report the accuracy per preposition.

Results Their results are summarized in table 6.5. The neural scoring model with one-hot vectors for words and POS tags (NN1) achieves only moderate results and is outperformed by nearly 14% when replacing the one-hot vectors with word embeddings (NN2). Including the topological field tags (NN3) and auxiliary distributions (NN4, NN5) as additional features further improves the system's performance. The best result of 86.7% is achieved by including only auxiliary distributions calculated on the unambiguous triples (NN5), but the difference is insignificant (+0.2%).

The success of the above system is a combination of well-known techniques for PP attachment disambiguation for English (auxiliary distributions, embeddings), combined with language-specific knowledge (topological fields) and modeling (neural

[1]The data size in the paper (43,906) is mistakenly from an older version of the data.

networks). However, there are still some shortcomings that need to be addressed. First, no results are reported regarding the relative performance of the system as compared to a strong baseline parser. Second, the fact that system performance is reported on (and, crucially, relies on) gold standard features makes it less attractive in comparison to parsing systems that are able to perform well also with predicted features. This is the motivation behind our work. In the remainder of the chapter, we will address these issues by evaluating PP attachment disambiguation systems in a truly realistic setting and assessing their contribution in comparison to a full parsing setup.

6.3 Evaluating PP Attachment in a Realistic Setup

The goal of our experiments is to overcome the limitations of previous work outlined above. In particular, we now experiment with PP attachment disambiguation in a more realistic scenario where gold information like POS tags and topological field structures is not available. Our reference systems are the dependency parser with biaffine classifiers (Dozat and Manning, 2017) which is among the best systems for parsing German, and the PP attachment disambiguation system from de Kok et al. (2017b) which has already been introduced in section 6.2.2.

6.3.1 Reproducing PP Attachment Results of de Kok et al. (2017b)

We first try to reproduce the results of de Kok et al. (2017b) described in section 6.2, using our re-implementation of their disambiguation system. Since our goal is to compare the performance of the system to that of a parser, we first focus on the setting without external knowledge except for word and POS tag embeddings (i.e., setting NN3 in table 6.5).

We report results for the PP attachment data set (de Kok et al., 2017a) extracted from the TüBa-D/Z, using the same train/test split as in de Kok et al. (2017b). From the 29,033 instances that have been removed from the data set for training a parser in order to produce auxiliary distributions (de Kok et al., 2017b), we randomly select a development set of size 8,649, since we do not use the auxiliary distribution features and it is more straightforward to train the system with a development set. The size of each set is given in table 6.6. The input to the system is the same as described in section 6.2.2 excluding the association scores. However, even with the same word and POS tag embeddings, we could not reproduce the results in de Kok et al. (2017b). Our re-implemented system achieves an accuracy of only 81.1%, 2.7% below the reported result.

Data Set	Size
Train	35,076
Dev	8,649
Test	8,769

Table 6.6: Statistics of the PP attachment disambiguation data

We observe that the cross-entropy loss used to train the system might not be the most suitable choice, since it optimizes the score of each candidate independently. Instead, we argue that the system should benefit more if it scores each candidate with respect to the others. This observation leads us to replace the cross-entropy loss with the hinge loss which tunes the score of the correct head higher than that of the incorrect one. Given the input features x, the correct head y, and an incorrect head h. Θ and $s(\cdot)$ are the system parameters and the score returned by the system. The loss of h is calculated as:

$$L(y,h) = \max(0, s(x,h,\Theta) + 1 - s(x,y,\Theta)) \tag{6.11}$$

The final loss is the average loss of all incorrect heads:

$$\begin{aligned}\mathcal{L}(\Theta) &= \frac{1}{|\mathcal{D}|}\sum_{x,y\in\mathcal{D}}\frac{1}{|\mathbf{neg}(x)|}\sum_{h\in\mathbf{neg}(x)} L(y,h) &\quad (6.12)\\ &= \frac{1}{|\mathcal{D}|}\sum_{x,y\in\mathcal{D}}\frac{1}{|\mathbf{neg}(x)|}\sum_{h\in\mathbf{neg}(x)} \max(0, s(x,h,\Theta) + 1 - s(x,y,\Theta)) &\quad (6.13)\end{aligned}$$

where $\mathcal{D}$ is the training data and $\mathbf{neg}(x)$ is the list of incorrect candidate heads of input x.

By changing the loss function and increasing the size of the hidden layer to 1000, the accuracy of our system increases to 84.1%, which is in the same range as the published result of de Kok et al. (2017b). The differences between our implementation and the original system are summarized in table 6.7. We call this setting **PP-Rep**.

For computing the auxiliary distributions, we use articles from the *taz* newspaper from 1986 to 1999 (11.5 million sentences, 204.4 million tokens), which is actually a subset of the data used to compute the auxiliary distributions in de Kok et al. (2017b). We parse the corpus with the graph-based parser from the MATE tools[2] (Bohnet, 2010) trained on the German data set from the CoNLL 2009 Shared Task (Hajič et al., 2009). We keep ambiguous and unambiguous triples and calculate 5 association scores as described in section 6.2.2 (similar to the setting NN4 in table 6.5). With the

[2]https://code.google.com/p/mate-tools

Hyperparameter	de Kok et al. (2017b)	Our re-implementation
Hidden dim.	100	1000
Loss	Cross-entropy	Hinge
Input dropout	0.2	0
Hidden dropout	0.05	0
Batch normalization	Yes	No

Table 6.7: The differences between the PP attachment disambiguation system from de Kok et al. (2017b) and our re-implementation

same hyperparameters as PP-REP, our system with auxiliary distributions achieves an accuracy of 86.8%, slightly higher than the same setup (NN4) (+0.3%) and the best reported result (NN5) (+0.1%) from de Kok et al. (2017b). We refer to this setting as **PP-REP-AUX**. The summary of our reproduced results is in table 6.5.

6.3.2 Upper Bounds for PP Attachment Disambiguation *without* Gold Information

After our successful reproduction of previous work, we now proceed to disambiguate PP attachments in a setup that *does not rely on gold POS and topological field tags* and compare its performance with that of the reference neural dependency parser from Dozat and Manning (2017). In order to do so, the instances of the same data used in section 6.3.1 (in the form of preposition, prepositional object, and a list of candidate heads and features) have to be mapped back to the original dependency trees to enable a comparison with full parsing. Unfortunately, the process of mapping the triples back to the corresponding treebank trees is not straightforward. First, the format of the PP attachment data set (de Kok et al., 2017a) does not provide the information needed to trace back the training instances to the original trees as it lacks the original structural information (e.g., the position of the words in each instance). Thus, one instance can be ambiguously mapped to different trees (or different positions in the same tree). Second, the train/test split is done by randomly selecting PP *instances*, not *sentences*. Therefore, when being mapped back, there are trees that appear in both the train and test splits. Of course, it would be easy to remove those trees from the training set in order to create non-overlapping sets, but then the results would not be comparable to the parsing results, as those are obtained on a different data set.

For those reasons, we decide to experiment on the German data set from the

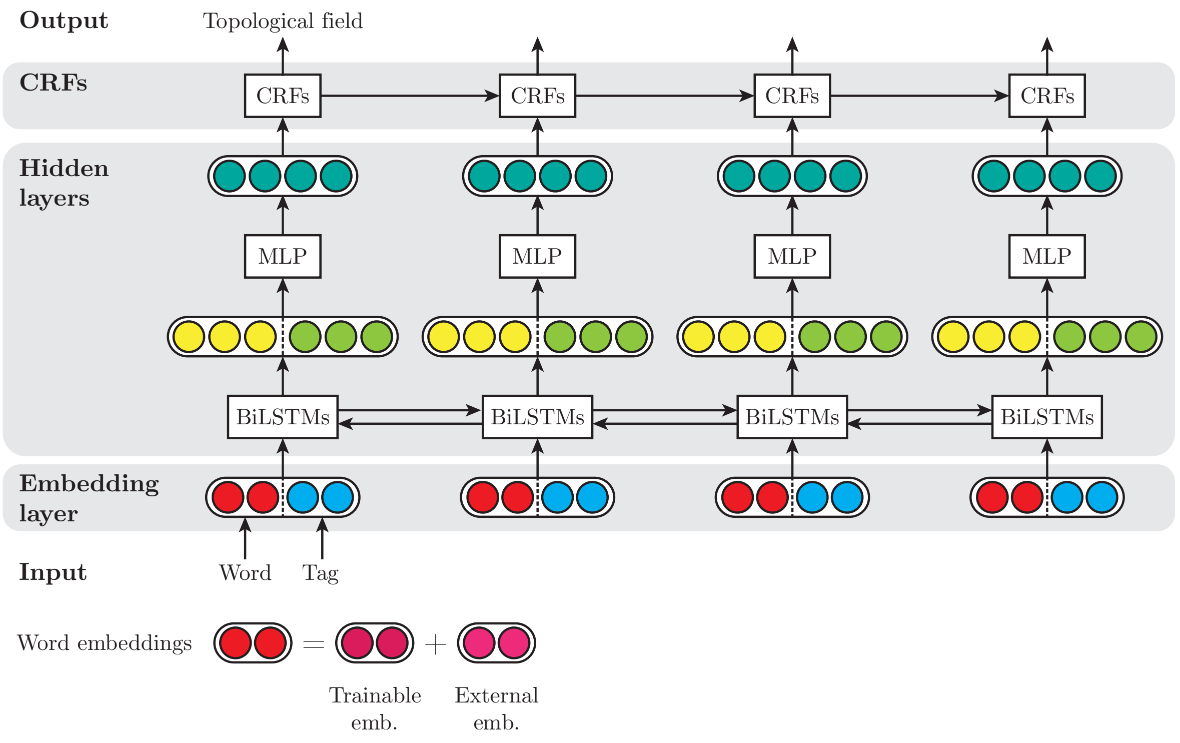


Figure 6.4: The architecture of the topological field labeler

SPMRL 2014 Shared Task (Seddah et al., 2014). The treebank consists of 50,000 sentences of German newspaper text, which is slightly larger than the PP attachment data set used in de Kok et al. (2017b) but does not contain gold information on topological fields. Thus, in the next step, we will predict topological fields for this data set.

Reference parser We re-implement the parser of Dozat and Manning (2017) and train it with default hyperparameters. The pre-trained embeddings are 100-dimensional dependency based word embeddings (Levy and Goldberg, 2014) trained on the SdeWaC corpus (Faaß and Eckart, 2013) with a cutoff frequency of 20 for both words and contexts, and the number of negative samples of 15. Our model achieves a 93.65% unlabeled attachment score (UAS) and a 92.22% labeled attachment score (LAS) on the German SPMRL test set using the predicted POS described below.

Predicting POS and Topological Fields Following de Kok and Hinrichs (2016), we model topological field prediction as a sequence labeling task. Rather than predicting deep topological field structure (which are recursive tree structures with potentially nested layers of annotation), we flatten the trees and only annotate each word with

TIGER	TüBa-D/Z	Examples	
PROAV	PROP	dabei, daraufhin, damit, davon...	different names
APPR	KOKOM	**als** einzige, **als** US-Präsidenten	preposition vs. comparative particle
PIAT	PIDAT	in **jedem** Einzelfall	TIGER doesn't distinguish between PIAT and PIDAT and uses PIAT only; PIDAT is only used in TüBa-D/Z
ADJD	ADV	**Wahrscheinlich** mache ich...	both tags exist in both treebanks but TIGER tags adverbial uses of *wahrscheinlich* as ADJD and TüBa-D/Z as ADV

Table 6.8: Differences between the tag set of the Tiger and the TüBa-D/Z treebanks.

its *nearest* topological field tag from the tree hierarchy, as in figure 6.3. Although this method leads to a loss in information when the sentence contains nested fields, it corresponds with the features used in de Kok et al. (2017b). Our topological field labeler is a neural network system with two bidirectional LSTM layers and a conditional random field (CRF) decoder. The architecture of the labeler is illustrated in figure 6.4. Details about the hyperparameters can be seen in appendix A.5.

The labeler is trained on the Universal Dependencies (UD) version of the TüBa-D/Z (Çöltekin et al., 2017), which was randomly split into train/dev/test sets with a size of 94,210/5,230/5,347 trees. The data are preprocessed to match the format in the SPMRL data: open (`(`, `[`, ») and closing brackets (`)`, `]`, «) are replaced with ‟ and ”, preposition contractions (which have been split into definite articles and the prepositions) are rejoined into single tokens.

Because the SPMRL and the TüBa-D/Z corpora use slightly different POS schemeta[3], we thus use MarMoT (Mueller et al., 2013) to train a POS tagger on the training set of the SPMRL data set[4] and re-tag the TüBa-D/Z. Table 6.9 shows the performance of the topological field labeler for using gold POS tags and predicted SPMRL POS tags. Although using predicted POS tags decreases the accuracy (per word) by only ∼1%, the whole sentence accuracy dramatically drops by 4%.

[3]Both use the STTS but interpret the guidelines differently. The most important differences are summarized in table 6.8.

[4]The POS tagger achieves 97.17% on the SPMRL test set.

	Gold POS		Predicted POS	
Accuracy	Dev	Test	Dev	Test
POS	100.00	100.00		
Topological field	96.91	96.88	96.10	95.74
Sentence	86.92	86.78	82.58	82.72

Table 6.9: Topological field labeling results on the TüBa-D/Z corpus

Data Set	Accuracy	Sentence accuracy
Train	98.06	74.84
Dev	98.23	79.50
Test	97.65	70.62
All	98.03	74.89

Table 6.10: Accuracy of 10-way jackknifing POS tags on the German data set of the SPMRL 2014 corpus using MarMoT

For the SPMRL data, we use MarMoT to assign the POS tags with 10-way jackknifing[5] and predict the topological field tags using the topological field labeler.

Candidate extraction In our first experiment, we study the effect of using *automatically predicted* information for candidate extraction. We replace the gold POS and topological fields with the predicted ones and calculate the upper bound accuracy. We follow the rules described in de Kok et al. (2017a) to extract the nominal candidate heads for each preposition[6] but use the gold tree to find the verbal candidate (the main verb) rather than using both topological fields and gold trees. In summary:

- Prepositions are determined using the predicted POS tags.
- Prepositional objects are determined using the gold trees.
- Nominal candidates are extracted based on the predicted POS tags and topological field tags.
- Verbal candidates are extracted based on gold tree information.

[5]The accuracy of the POS tagger on the whole SPMRL data set is 98.03% (see table 6.10).

[6]We follow de Kok et al. (2017a) and consider *adpositions* (both *prepositions* and *postpositions*) in our experiments.

	POS	TF	P	O	N	V
de Kok et al. (2017a)	gold	gold	gold	gold	gold	gold
Experiment 1 (§6.3.2)	pred	pred	pred	gold	pred	gold
Experiment 2 (§6.3.3)	pred	pred	pred	gold/pred	pred	pred

Table 6.11: Gold/predicted features used in PP attachment experiments. POS: POS tags, TF: topological fields, P: prepositions, O: prepositional objects, N: nominal candidate heads, V: verbal candidate heads.

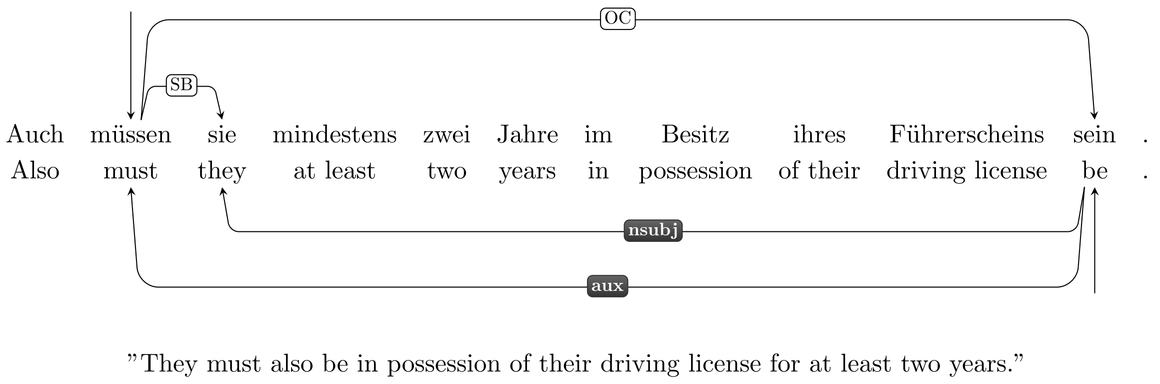

OC
SB
Auch müssen sie mindestens zwei Jahre im Besitz ihres Führerscheins sein .
Also must they at least two years in possession of their driving license be .
nsubj
aux
"They must also be in possession of their driving license for at least two years."

Figure 6.5: Differences in verb and core argument attachments between the SPMRL 2014 (above, white) and the TüBa D/Z (below, dark) corpora

Table 6.11 sums up the differences between the features used in de Kok et al. (2017b) and in our experiment. One of the main differences in the annotation schemes between the TüBa-D/Z corpus and the SPMRL data (see figure 6.5) is that the auxiliary verb in TüBa-D/Z corpus is attached to the main (non-auxiliary) verb with label *aux*, while in the SPMRL data, the main verb is connected to the auxiliary verb with label *OC*.

Upper bound for PP attachment We call the set of all gold standard prepositions in the SPMRL test data **PP-SPMRL** (9,273 instances). On the test set of the SPMRL data with predicted POS tags and topological fields, the extraction algorithm finds at least one candidate for 91.66% of the prepositions. The failure cases, called *non-attachments* (see figure 6.6), are cases where the ambiguity cannot be recognized, or not covered by the extraction rules (see figure 6.6). These cases include:

- instances where the preposition does not have an object (for instance, *bis zum* (by) construction, see figure 6.7), or
- instances where the preposition is tagged with a wrong topological field outside

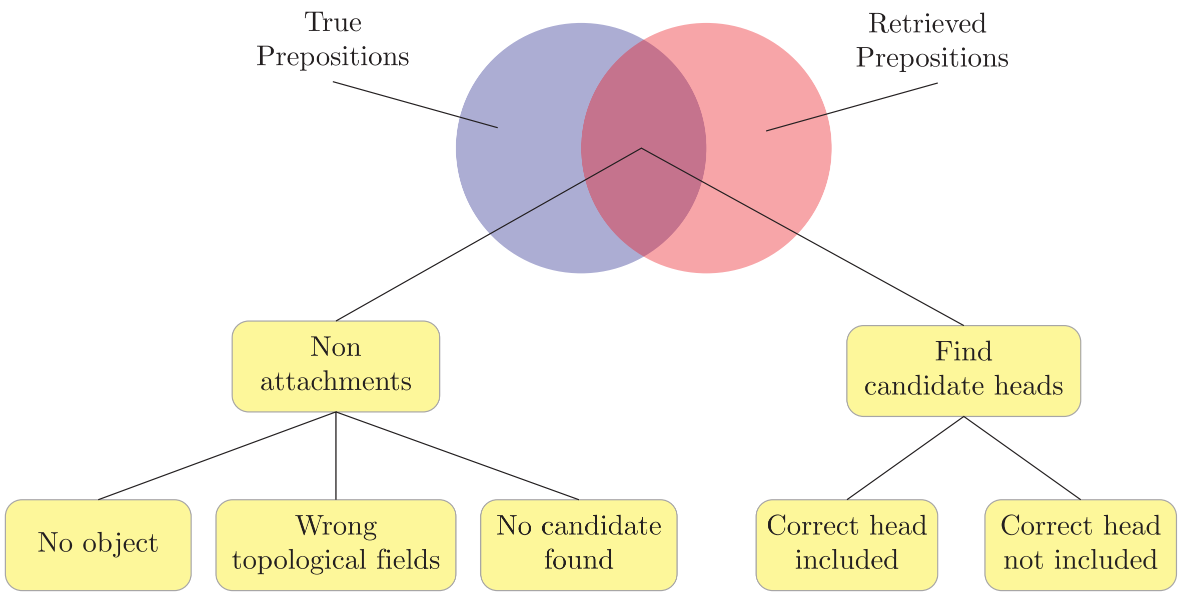


Figure 6.6: Extracting a PP attachment disambiguation data set with predicted information

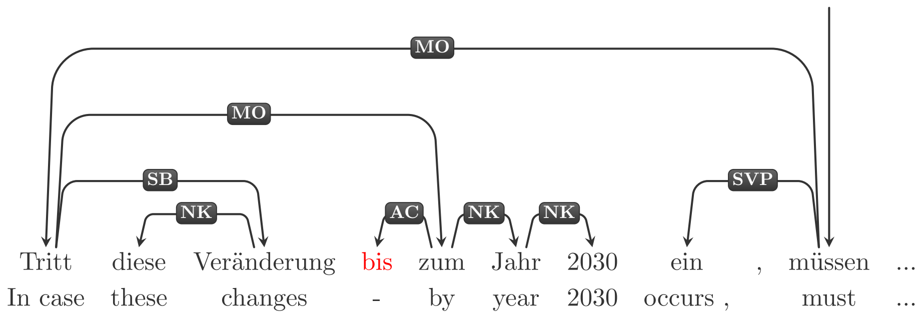

MO
MO
SB
NK
AC
NK
NK
SVP
Tritt diese Veränderung bis zum Jahr 2030 ein , müssen ...
In case these changes - by year 2030 occurs , must ...

Figure 6.7: A sentence from the SPMRL 2014 test set contains preposition *bis* without an object

the VF, MF, or NF (therefore, is not covered by the extraction rules), or

- instances where the extraction rules cannot find any candidate head because there are errors in the boundaries of the current topological field level so that nominal candidate heads cannot be found (verbal candidate heads are undoubtedly included because we use the gold tree to resolve them).

The correct heads are detected in only 82.73% of all cases. This is not only caused by errors in the prediction of topological fields (that make the correct nominal head not included) but also because the algorithm only considers nouns and verbs as possible candidates.[7] This means that the upper bound for recall for PP attachment

[7]For 9.89% of the prepositions in the SPMRL test set, the head is neither a noun nor a verb.

disambiguation is 82.73% on the test set of the SPMRL data. In comparison, the reference parser achieves 86.17% accuracy for predicting the head of each preposition on the same data set.

If we only consider 8,360 instances where the gold head of a preposition is either a noun or a verb (we call this set **PP-SPMRL-NV**), the upper bound recall of the disambiguation system increases to 91.56%, whereas the accuracy of the reference parser is 87.21%. Our experiment shows that by limiting head candidates to nouns and verbs only and using predicted POS tags and topological fields, the disambiguation system always performs worse than the parser when evaluating on *all* prepositions. In the next experiment, we will see if the disambiguation system has any advantage over the parser when considering only prepositions with nominal or verbal heads.

6.3.3 Real-World Evaluation of PP Attachment Disambiguation and PP Reattachment

After having established the upper bound performance that we can expect in a realistic scenario, we now assess the performance of the PP attachment disambiguation approach on the output of a strong parser. That is, we are interested in whether the PP attachment disambiguation system can help to improve the performance of the reference parser. Following Atterer and Schütze (2007), we now replace the gold trees used in the previous experiment (section 6.3.2) with the ones predicted by the parser.

The only difference to the previous procedure to extract candidate heads (section 6.3.2) is that we now rely on the parser's predictions also for finding verbal candidates and prepositional objects, instead of using gold information as before. More specifically, at *test* time:

- Prepositions are identified using the predicted POS tags.
- Prepositional objects are determined using the *predicted* parse trees.
- Nominal candidates are extracted based on the predicted POS and topological field tags.
- Verbal candidates are extracted using the *predicted* parse trees.

Thus, the gold trees are only used to assign the output labels (indicating the candidate is the correct or incorrect head) for extracted instances. For the train and dev sets, in contrast, only correct instances (prepositions with nominal or verbal heads according to gold information)) are considered. All other cases are filtered out as the extraction

Data Set	Size
Train	45,129
Dev	5,463
Test	8,516

Table 6.12: Statistics of the PP attachment disambiguation data sets extracted from the German SPMRL data

rules of de Kok et al. (2017b) are restricted to finding nominal and verbal candidates, hence the disambiguation system is only able to select the correct head among nouns and verbs. We further add the correct head to the candidate set in the training and development data if the extraction algorithm fails to include it. The sizes of the data after extraction are reported in table 6.12. In addition to the non-attachment cases described in section 6.3.2, in this experiment, the correct verbal head may not be found because of the errors in the predicted trees (e.g., a tree has a loop). The types of features used in this experiment are summarized in table 6.11.

In comparison to de Kok et al. (2017a), our data set addresses a more realistic scenario in which the candidate heads for the PP are chosen based on *predicted information*, thus the accuracy for PP attachment is bound by the errors from all steps in the pipeline. We train the same system used to reproduce the results of de Kok et al. (2017b) (section 6.3.1) on our newly created data set for PP attachment disambiguation. Note that the data we used to train the parser for association scores (section 6.3.1) are the same as the training data of the German SPMRL data set (both are based on the TIGER Treebank).

Evaluation metrics for PP attachment disambiguation We follow Agirre et al. (2008) and report *precision*, *recall* and F1 for PP attachment disambiguation. An instance is considered *correct* if:

- its preposition is also a preposition in the gold standard, and
- the system identified the correct head for the preposition.

An instance is considered *incorrect* if:

- its preposition is also a preposition in the gold standard, but
- the system assigns the incorrect head for the preposition.

Instances, of which prepositions are not included in the gold standard, are discarded. *Precision* is calculated as the number of correct instances divided by the number of correct and incorrect instances (but ignoring the discarded instances):

$$\text{precision} = \frac{|\{\text{correct instances}\}|}{|\{\text{correct instances}\} \cup \{\text{incorrect instances}\}|} \tag{6.14}$$

while *recall* is measured as the number of correct instances divided by the number of prepositions in the gold data:

$$\text{recall} = \frac{|\{\text{correct instances}\}|}{|\{\text{gold instances}\}|} \tag{6.15}$$

Non-attachment cases are discarded from the evaluation, similar to the `NA-disc(ard)` metric from Atterer and Schütze (2007).

Evaluation metrics for PP reattachment In addition to the PP attachment metrics described above, we also report combined results for the reference parser and the PP attachment disambiguation system (i.e., we *reattach* the PPs predicted by the parser). In the parser output, the head of a preposition is replaced with the one predicted by the PP attachment system in two different ways:

- `all`: We replace the head of a preposition in the parser output with the one predicted by the disambiguation system for *all* prepositions.
- `N&V`: We only replace the head of a preposition if the current head predicted by the parser is a noun or a verb.

The results of PP reattachment are reported as accuracy for PP attachment, and unlabeled attachment score (UAS) for the whole parse tree. Note that accuracy in PP reattachment corresponds to *all* precision, recall, and F1 for PP attachment disambiguation (because the parser predicts a head for all words, the sets for gold and retrieved prepositions are the same).

Results Table 6.13 shows the performance for PP attachment disambiguation and PP reattachment for different settings.[8] Using the parser's predictions to extract verbal candidate heads (Exp.2) instead of gold information (Exp.1) further reduces the upper bound recall for PP attachment in section 6.3.2. Neither PP-REP nor PP-REP-AUX could surpass the parser in choosing the correct head for the prepositions. Even when combining these systems with the parser and reattaching the PPs, the accuracies are still behind that of the parser on its own. By restricting the reattachment mechanism to those cases where the head determined by the parser is either a noun

[8]Our PP-REP and PP-REP-AUX systems were trained on data sets with gold prepositional objects, but we found no difference in performance when training them with predicted objects.

	PP Attachment						PP Reattachment		
	PP-SPMRL			PP-SPMRL-NV			PP-SPMRL	PP-SPMRL-NV	Parsing
System	P	R	F1	P	R	F1	Accuracy	Accuracy	UAS
Parser	86.29	85.73	86.01	87.33	86.87	87.10	86.17	87.21	93.65
Exp.1 (§6.3.2)									
Upper bound	100	82.73	90.55	100	91.56	95.59	–	–	–
Exp.2 (§6.3.3)									
Upper bound	100	80.45	89.17	100	89.03	94.20	–	–	–
PP-REP	77.76	71.22	71.45	84.59	78.80	81.60	–	–	–
`all`	–	–	–	–	–	–	82.68	85.84	93.29
`N&V`	–	–	–	–	–	–	84.91	85.81	93.52
PP-REP-AUX	79.52	72.84	76.03	86.52	80.60	83.45	–	–	–
`all`	–	–	–	–	–	–	83.46	86.63	93.37
`N&V`	–	–	–	–	–	–	85.59	86.58	93.59

Table 6.13: PP attachment disambiguation and PP reattachment results on the German SPMRL test set.

Error	Count
Topological field of the preposition is not VF, MF, NF	620
Cannot find any candidate head	5
Preposition has no object	95
Correct head not included	1,033
Retrieved	9,302

Table 6.14: Types of errors in PP candidate head extraction. The error is categorized as the first error encountered from top to bottom. Error categories are not exclusive.

or a verb (N&V), the accuracy for PP reattachment on the PP-SPMRL set increases by 2%, as this reduces the risk of reattaching the preposition to a nominal or verbal head when the head actually belongs to another part of speech. Adding auxiliary distribution scores to the system (PP-REP-AUX) consistently improves the performance for all settings (over PP-REP).

Our experiments show that an independent disambiguation system has no advantage over a full parser in determining the correct head for a preposition when tested in a realistic scenario. First, despite its promising capability in experiments with gold information, the upper bound of the system has already been limited by the error propagation when extracting the candidates based on predicted tags and trees. Some error types in the candidate extraction process are listed in table 6.14. Second, restricting the coverage of the system by considering only nouns and verbs as possible heads results in a high number of misclassified instances. In contrast, neural network dependency parsers are trained end-to-end on predicted information, thus reducing error propagation. They also make better use of the data by training a common classifier for all head-dependent types, an efficient way to cope with data sparseness.

6.4 PP Attachment without Restrictions

In the previous section, we have shown that techniques to extract plausible head candidates for German PP attachment disambiguation based on topological fields decrease the upper bound of the system, either by reducing coverage or by error propagation when combining the candidate extraction rules with predicted information at test time. In this section, we will thus consider *all words in a sentence* as possible heads instead of restricting candidate selection to certain nouns and verbs, and focus

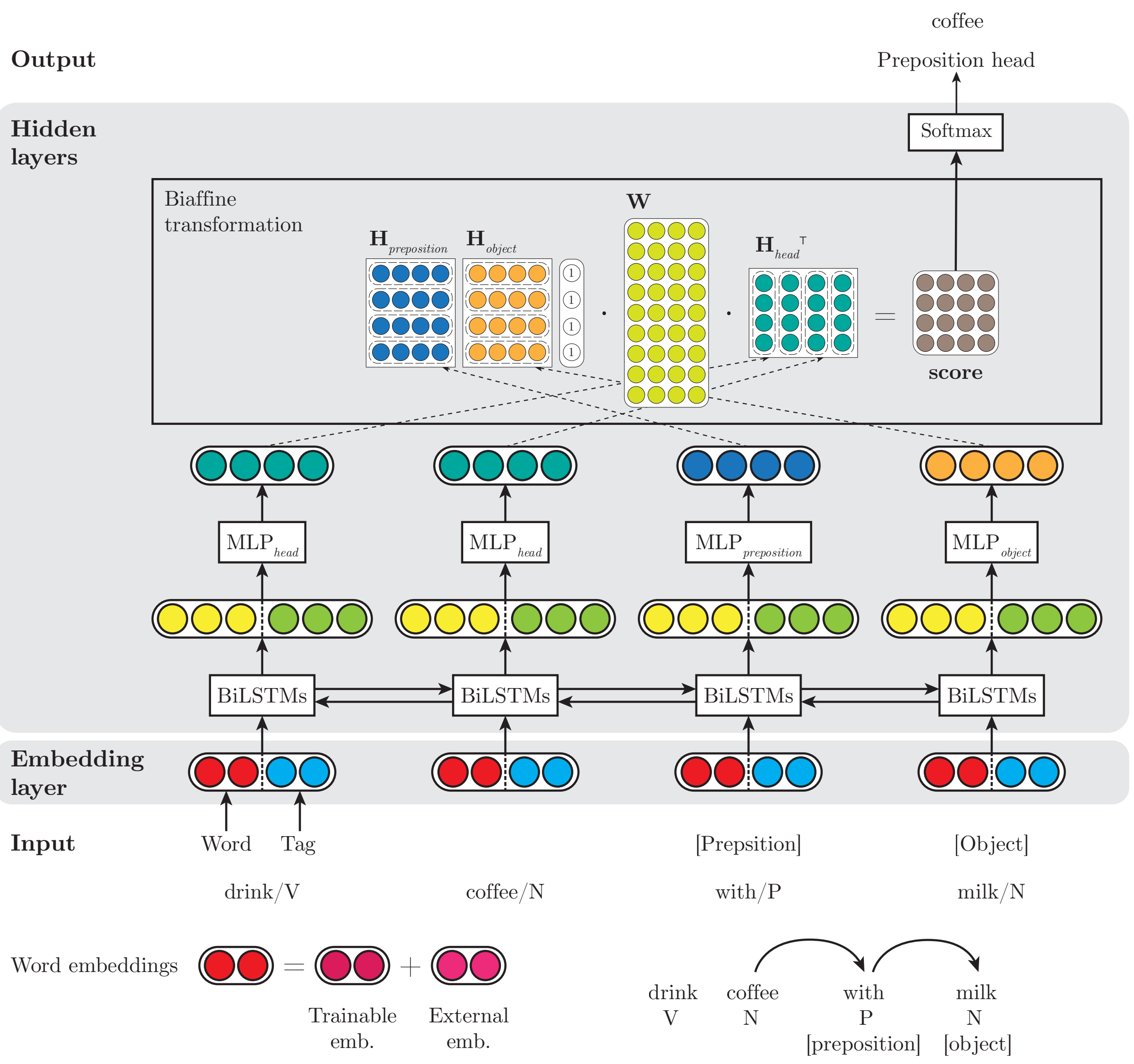


Figure 6.8: PP-BIAFFINE: PP attachment disambiguation system with biaffine transformations

on techniques that can cope with data sparseness and deal with noisy input data in real-world scenarios.

PP attachment disambiguation with biaffine transformations Our first model is **PP-BIAFFINE**, a PP attachment disambiguation system with biaffine transformations similar to the reference parser of Dozat and Manning (2017). The architect of the system is illustrated in figure 6.8. The input to the system consists of words, POS tags, and the position of the preposition and its object. Prepositions are identified based on predicted POS tags, and their objects are extracted based on gold standard dependency trees for training and predicted parse trees at test time.[9] Words and tags are converted to embeddings, and the word embeddings are the sum of the randomly initialized embeddings and the pre-trained word embeddings. The input representation of a token is the concatenate of the corresponding word and tag embeddings:

$$\mathbf{e}_i = [\mathbf{e}_i^{\text{word}}; \mathbf{e}_i^{\text{tag}}] \tag{6.16}$$

Tokens in a sentence are encoded using several bidirectional LSTM layers, and the outputs are projected using 3 non-linear transformation MLP$_H$, MLP$_P$, MLP$_O$ corresponding to head, preposition and object representations:

$$\mathbf{h}_i = \text{BiLSTMs}(\mathbf{e}_{1:n}, i) \tag{6.17}$$
$$\mathbf{h}_i^H = \text{MLP}_H(\mathbf{h}_i) \tag{6.18}$$
$$\mathbf{h}_i^P = \text{MLP}_P(\mathbf{h}_i) \tag{6.19}$$
$$\mathbf{h}_i^O = \text{MLP}_O(\mathbf{h}_i) \tag{6.20}$$

The score for word h being the head of preposition p with object o is computed using a biaffine transformation:

$$s_{\text{biaffine}}(h, p, o) = \mathbf{h}_p^{P\top}\mathbf{W}_1\mathbf{h}_h^H + \mathbf{h}_o^{O\top}\mathbf{W}_2\mathbf{h}_h^H + \mathbf{w}^\top\mathbf{h}_h^H \tag{6.21}$$

where $\mathbf{W}_1$ and $\mathbf{W}_2$ are weight matrices, $\mathbf{w}$ is a weight vector. The system is trained with a cross-entropy loss function. At test time, the system returns the word with the highest score as the predicted head of the PP.

When using topological field tags (+topo), we also convert them into embeddings and concatenate them with word and tag embeddings to form the token representation:

$$\mathbf{e}_i = [\mathbf{e}_i^{\text{word}}; \mathbf{e}_i^{\text{tag}}; \mathbf{e}_i^{\text{topo}}] \tag{6.22}$$

In settings with auxiliary distribution (+aux), the 5 auxiliary scores s_{aux} are combined with the biaffine scores s_{biaffine} using a linear combination:

$$s(h, p, o) = s_{\text{biaffine}}(h, p, o) + s_{\text{aux}}(h, p, o)\mathbf{w_s}^\top \tag{6.23}$$

The pre-trained embeddings for words are the same as used to train the reference parser (section 6.3.2). Information about hyperparameters and training details of the model can be found in appendix A.6.

We evaluate the potential of PP-BIAFFINE on both tasks, PP attachment disambiguation and PP reattachment (table 6.15). For PP attachment, PP-BIAFFINE

[9]We also trained the system on predicted prepositional objects but using gold objects produced slightly higher results.

outperforms PP-Rep and PP-Rep-Aux on the set of all prepositions (PP-spmrl) by a large margin. On the set with nominal and verbal heads (PP-spmrl-nv), the precision for PP-Biaffine +topo and PP-Biaffine +topo, +aux is similar to the one for PP-Rep and PP-Rep-Aux, but recall is much higher. The main reason for this is that PP-Biaffine considers all words as head candidates, while the candidate extraction rules can miss a correct attachment site because of errors in the input. This shows that not restricting the candidate set can lead to *better* performance for PP attachment disambiguation. However, the PP-Biaffine system is still not able to outperform the reference parser. Using PP-Biaffine to *reattach* the PPs predicted by the parser results in lower scores in comparison to using PP-Rep and PP-Rep-Aux for reattachment. We hypothesize that the performance of our PP attachment systems is still worse than that of the parser, which means that reattaching more PPs (due to the higher recall of PP-Biaffine) only lowers results.

PP attachment disambiguation with contextualized word embeddings Although having similar architectures, the performance of PP-Biaffine is still behind that of the full parser. The main reason for this is that the PP attachment disambiguation system has less training data: it is only trained on PP attachments while the parser trains a joint classifier for all attachment types. Instead of using more data, we propose to improve PP-Biaffine by *transfer learning* using BERT (Devlin et al., 2019). BERT is a language model based on multi-layer bidirectional Transformers (Vaswani et al., 2017) that are trained to be sensitive to positional context information, resulting in embeddings that represent *contextualized* word information (*contextualized word embeddings*) (see section 2.2.4). Devlin et al. (2019) have shown that the BERT$_{\text{LARGE}}$ model (with 340M parameters) achieves state-of-the-art results on a wide range of NLP tasks.

We now replace the pre-trained word embeddings in PP-Biaffine with embeddings provided by the BERT$_{\text{BASE}}$ Multilingual Cased model.[10] The rest of the system remains the same. We call this model PP-Biaffine+BERT. We do not fine-tune BERT on our data, as our experiments showed that this decreases results over simply using the pre-trained word embeddings. The performance of PP-Biaffine+BERT is shown in table 6.15. With the addition of BERT, our model outperforms both PP-Biaffine and the reference parser on both tasks, PP attachment disambiguation and PP reattachment. Adding topological field information (PP-Biaffine+BERT, +topo) results in slightly worse results, while the addition of both topological fields and auxiliary distributions (PP-Biaffine+BERT, +topo, +aux) outperforms all previous

[10]The BERT$_{\text{BASE}}$ Multilingual Cased (110M parameters) was trained on cased text from Wikipedia for 104 languages.

	PP Attachment						PP Reattachment		Parsing
	PP-SPMRL			PP-SPMRL-NV			PP-SPMRL	PP-SPMRL-NV	
System	P	R	F1	P	R	F1	Accuracy	Accuracy	UAS
Parser	86.29	85.73	86.01	87.33	86.87	87.10	86.17	87.21	93.65
PP-REP (N&V)	77.76	71.22	71.45	84.59	78.80	81.60	84.91	85.81	93.52
PP-REP-AUX (N&V)	79.52	72.84	76.03	86.52	80.60	83.45	85.59	86.58	93.59
PP-BIAFFINE	83.26	82.72	82.99	84.68	84.23	84.46	83.17	84.58	93.32
+topo	83.48	82.94	83.21	84.79	84.34	84.56	83.38	84.69	93.35
+topo,+aux	85.02	84.47	84.75	86.28	85.83	86.05	84.91	86.17	93.50
PP-BIAFFINE+BERT	87.06	86.50	86.78	88.13	87.67	87.90	86.94	88.01	93.71
+topo	86.87	86.30	86.58	88.07	87.61	87.84	86.75	87.95	93.70
+topo,+aux	87.32	86.75	87.03	88.42	87.95	88.19	87.19	88.30	93.74
Parser+BERT	88.40	87.82	88.11	89.45	88.98	89.22	88.35	89.43	94.43

Table 6.15: PP attachment disambiguation and PP reattachment results on the German SPMRL test set

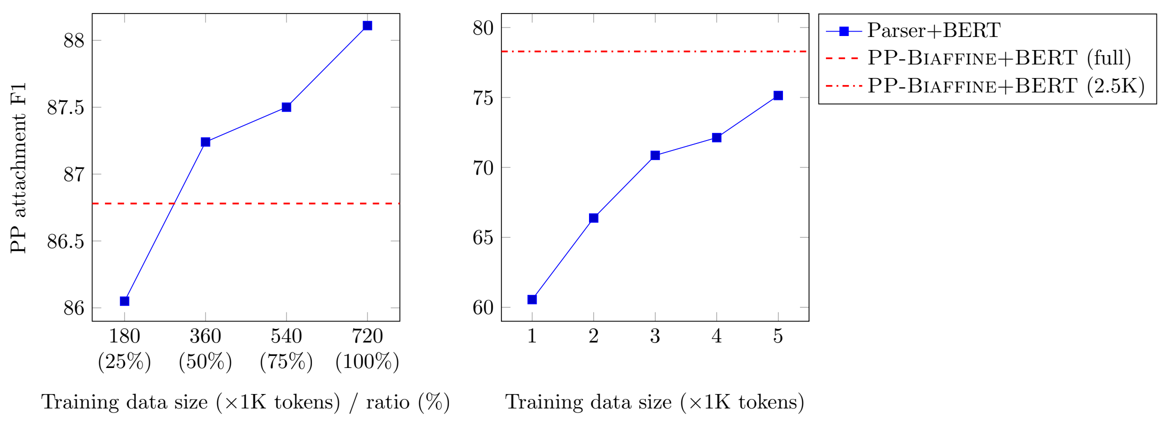


Figure 6.9: PP attachment disambiguation performance of the reference parser when reducing the size of the training data. Each data point is the average of three different reduced data sets of the same size.

models so far. However, the improvement we get is lower than the one we obtained when adding auxiliary scores to PP-BIAFFINE (table 6.13). This suggests that the information BERT learns from raw text is similar to the one provided by the auxiliary distribution scores.

Parsing with contextualized word embeddings In a similar fashion, we can replace the pre-trained word embeddings in the reference parser with BERT to further improve its performance. Using the same BERT$_{\text{BASE}}$ Multilingual Cased model without fine-tuning, parsing accuracy increases for both PP attachment (+∼2%) and parsing in general (+0.78%) (Parser+BERT in table 6.15), and excels the performance of PP-BIAFFINE+BERT, +topo, +aux by ∼1%. Again, the shared classifier mechanism has an advantage over the system dedicated to predicting PP attachments only.

Our experiments suggest that parsing systems are in general superior to systems specialized for PP attachment disambiguation. This, however, is only true for high resource languages like English and German where we have enough training data to train a good parser. For low resource languages, on the other hand, acquiring more data for PP attachment disambiguation is much easier than getting more annotated full trees for parser training because PP attachment disambiguation systems only require input in form of triples of (head, preposition, PP object). Moreover, systems for PP attachment disambiguation can utilize data from different treebanks, even if they are based on different underlying linguistic theories as long as they agree on the attachment site of the prepositions.

We reduce the amount of data used to train the reference parser to see when the specialized system has an advantage over the parser. The German SPMRL training set contains 40K sentences and 720K tokens. In the first experiment, we create training data sets that are 25%, 50%, and 75% of the original size. The hyperparameters of the parser are kept the same as in previous experiments (section 6.3.2, section 6.3.3)[11]. In the second experiment, we simulate a low resource scenario by creating training data sets of sizes ranging from 1,000 to 5,000 tokens. We heuristically reduce the dimensions and number of layers of the reference parser. Likewise, we train the PP-BIAFFINE+BERT[12] system in low resource mode with only 2,500 triples. In both experiments, we use the same POS tags as in previous experiments although POS accuracy could be lower in a real low resourced languages scenario.

The results are illustrated in figure 6.9. PP-BIAFFINE+BERT trained on the *full* PP attachment data (73K triples, or 146K dependency relations) only outperforms the parser when reducing the training data for the parser to 25% (180K tokens/dependency relations). In the simulated low resource setting, PP-BIAFFINE+BERT trained on 2.5K triples (5K dependency relations) clearly outperforms the parser trained on 5K tokens. The experiments confirm that with the same amount of data annotation, the PP attachment disambiguation system has an advantage over the parser. As the creation of a PP attachment data set consisting of triples is less expensive than annotating full syntax trees, we believe this could be a way to improve PP attachment accuracy for low resource languages.

6.5 Summary

In this chapter, we selected the task of PP attachment disambiguation to study the effect of semantic knowledge on dependency parsing. We presented an extensive study of the PP attachment disambiguation problem and proposed a new system that combines biaffine attention and pre-trained contextualized word embeddings. While our system outperforms recent work on German by a large margin, its performance is still inferior compared to that of a strong neural parser, thus questioning the approach of modeling PP attachment disambiguation as a separate task.

We showed that the lower results for the PP attachment system are caused by error propagation due to using predicted syntactic information for candidate extraction. In addition, the parser can make more efficient use of the training data. While the PP attachment disambiguation system is only trained on the PP attachment edges, the

[11]We follow the practice of the first ranked system in the CoNLL 2017 Shared Task on parsing UD treebanks Dozat et al. (2017).

[12]We assume that there is not enough data to train a good topological field predictor and a good parser for auxiliary distribution scores. For test data, we use the parser trained on 5K tokens to predict the prepositional objects.

parser makes use of all edge types in the tree to train a joint classifier that predicts the head of each word in a sentence. However, we argue that our system might still be useful for lower resourced languages where, due to a lack of training data, no strong parser is available. While our results are for German, the PP-BIAFFINE+BERT version of our system is language-agnostic and we expect that our findings will carry over to other languages. We leave this for future work.

CHAPTER 7

Sentence Level: Reranking Parse Trees

Neural models for dependency parsing have been a tremendous success, pushing state-of-the-art results for English on the WSJ benchmarking data set to over 94% LAS (Dozat and Manning, 2017). Most state-of-the-art parsers, however, are local and greedy and are thus expected to have problems finding the best *global* parse tree. This suggests that combining greedy, local parsing models with some mechanism that adds a global view on the data might increase parsing accuracies even further.

In this chapter, we look into dependency parsing at the **sentence** level and incorporate global information for dependency parsing via *reranking*. Different model architectures have been proposed for neural reranking of dependency parse trees (Le and Zuidema, 2014; Zhu et al., 2015; Zhou et al., 2016). Despite achieving modest or even substantial improvements over the baseline parser, however, all the systems above only report performance on English and Chinese data, both morphologically poor languages with a configurational word order and mostly projective tree structures. Therefore, in this work, we try to reproduce results for different reranking models from the literature on English data and compare them to results for German and Czech, two morphologically rich(er) languages (MRLs) with a high percentage of non-projective structures. In addition, we present a new discriminative reranking model based on graph convolutional networks (GCNs). Our GCN reranker outperforms the other rerankers on English and is also the only model able to obtain small improvements over the baseline parser on German and Czech while the other rerankers fail to beat the baselines. The improvements, however, are not significant and raise the question of what makes neural reranking of MRLs more difficult than reranking English or Chinese.

We analyze the differences in performance on the three languages and show that the reasons for this failure are due to the composition and quality of the k-best lists.

In particular, we show that the *gold tree ratio* in the English k-best list is much higher than for German and Czech, and that the trees in the English k-best list show a higher *variety*, thus making it easier for the reranker to distinguish between high- and low-quality trees.

For the rest of this chapter, we first review related work on reranking for neural dependency parsing (section 7.1) and describe different reranking models used in this chapter (section 7.2). After that, we reproduce reranking results for English and evaluate our new reranker on the English data (section 7.3). Finally, we test the different models on the two morphologically rich(er) languages and present the results of our evaluation and our analysis, before conclude in section 7.5.

7.1 Related Work

Reranking is a popular technique to improve the parsing performance on the output of a base parser. First, the top k candidate trees are generated by the base parser, then these trees are reranked using additional features not accessible to the base parser. This adds a more global and complete view of the trees, in contrast to the local and incomplete features used by the parser.

Discriminative rerankers have been a success story in constituency parsing (Collins and Koo, 2005; Charniak and Johnson, 2005). An important use case for reranking has been domain adaptation, and the combination of a discriminative, feature-rich reranker with a generative parsing model has been the reason behind the first successful self-training experiments (McClosky et al., 2006). A disadvantage of the traditional feature-rich rerankers is that the large number of potentially sparse features makes them prone to overfitting, and also reduces the efficiency of the systems. Neural rerankers offer a solution to that problem by learning dense, low-dimensional feature representations that are better at generalization, and so reduce the risk of overfitting.

Neural reranking The first neural reranker has been presented by Socher et al. (2013) for constituency parsing, based on a recursive neural network that processes the nodes in the parse tree bottom-up and learns dense feature presentations for the whole tree. This approach was adapted for dependency parsing by Le and Zuidema (2014). They presented an inside-outside recursive neural network (IORNN) which also adds a top-down flow of information to the model, in combination with the traditional bottom-up in the recursive neural network. Zhu et al. (2015) improve on previous work by proposing a recursive convolutional neural network (RCNN) architecture for reranking which can capture syntactic and semantic properties of words and phrases in the parse trees Through the use of convolution and pooling layers, the model is able to learn the best compositional representation for each node in the tree. The feature representations are then fed into a discriminative reranker that is applied to the k-best output of a base parser. The advantage of their model is that it can represent not only binary but also k-ary trees, as needed for dependency parsing (see section 7.2 for a more detailed description of the IORNN and RCNN models).

k-best vs. forest reranking There exist two different approaches to reranking for parsing: k-best reranking and forest reranking. In k-best reranking, the complete parse tree is encoded and presented to the reranker. A major disadvantage of k-best reranking is the limited scope of the k-best list which provides an upper bound for reranking performance. In contrast, a packed parse forest is a compact representation of exponentially many trees of which each node represents a deductive step. Forest reranking (Huang, 2008; Hayashi et al., 2011) approximately decodes the highest scored tree with both local and non-local features in a parse forest with cube pruning (Huang and Chiang, 2005).

In our work, we focus on neural reranking of a k-best list of parses generated by a base parsing system. Despite the advantage of forest reranking, the technique relies on generating parse forests. Unfortunately, we could not find any available parsers that are both non-projective and produce parse forests at the output to experiment on German.

7.2 Neural Reranking Models

In this section, we look into the reranking approach for dependency parsing. Two different neural reranking models are used for comparison: the *generative* inside-outside recursive neural networks (**IORNNs**) reranker (Le and Zuidema, 2014) and the *discriminative* reranker based on recurrent convolutional neural networks (**RCNNs**) (Zhu et al., 2015). In addition, we propose a new reranking model for dependency parsing that employs graph convolutional networks (**GCNs**) to encode the trees.

7.2.1 Generative Models

A generative reranking model scores a dependency structure by estimating its generation probability. The probability of generating a fragment of a dependency tree (e.g., a node) D depends on its dependency context C_D. For example, in model C of

Eisner (1996), the probability to generate a fragment $T(H)$ of dependency structure T rooted at H is defined as:

$$P(T(H)) = \prod_{l=1}^{L} P(H_l^L \mid C_{H_l^L}) \times \prod_{r=1}^{R} P(H_r^R \mid C_{H_r^R}) \tag{7.1}$$

where H^L and H^R are the left and right dependency of H. C_D is the dependency context in which node D is generated.

The amount of information used in C_D is called the *order* of the generative model. For instance:

- first-order, C_D^1: contains the head H of D
- second-order, C_D^2: contains H and the previous generated sibling S of D
- third-order, C_D^2: contains H, S, the sibling S' before S; or H, S, and the grand-head G

Ideally, we want to generate dependency fragment D based on ∞-order context C_D^∞, which includes all ancestors of D, their siblings, and all siblings of D. As the ∞-order counting model is impracticable due to data sparsity, Le and Zuidema (2014) propose the IORNN model to encode the context to generate each node in a dense vector.

IORNN The IORNN extends the idea of recursive neural networks (Socher et al., 2010) for constituent parsing where the *inner representation* of a node is computed bottom up. It also adds a second vector to each node, an *outer representation*, which is computed top down. The inner representation represents the content of the subtree at the current node, while the outer representation represents the context used to generate that node.

Given a tree $(p_2\ (p_1\ x\ y)\ z)$ (figure 7.1), the RCNN for constituent parsing computes the inner representation of p_1 based on its children x and y:

$$\mathbf{i}_{p_1} = f(\mathbf{W}_1^i \mathbf{i}_x + \mathbf{W}_2^i \mathbf{i}_y + \mathbf{b}^i) \tag{7.2}$$

The outer representation of p_1 takes into account the representations of its head p_2 and sibling z:

$$\mathbf{o}_{p_1} = g(\mathbf{W}_1^o \mathbf{o}_{p_2} + \mathbf{W}_2^o \mathbf{i}_z + \mathbf{b}^o) \tag{7.3}$$

$W_1^i, W_2^i, W_1^o, W_1^o$ are weight matrices; b^i and b^o are bias vectors; f and g are activation functions.

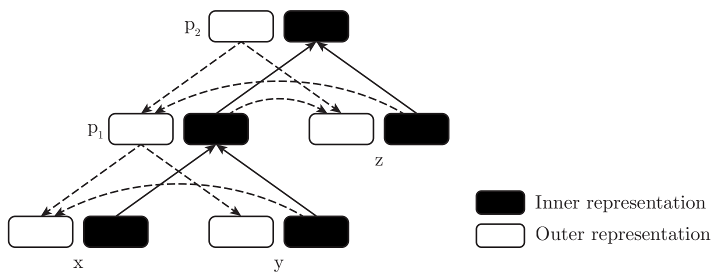


Figure 7.1: The IORNN for constituency trees (reproduced from Le and Zuidema (2014))

The model is further adapted to ∞-order dependency trees with *partial outer representation* that represents the partial context while generating dependents from left to right. First, the inner representation of each node h is calculated as:

$$\mathbf{i}_h = f(\mathbf{W}_w \mathbf{w}_h + \mathbf{W}_p \mathbf{p}_h + \mathbf{W}_c \mathbf{c}_h) \tag{7.4}$$

where $\mathbf{w}_h$ is the word vector of h, $\mathbf{p}_h$ and $\mathbf{c}_h$ are one-hot vectors representing the POS tag and capitalization feature of h. Then, the partial outer representation of child u of h is computed as:

$$\bar{\mathbf{o}}_u = f(\mathbf{W}_{hi}\mathbf{i}_h + \mathbf{W}_{ho}\mathbf{o}_h + b_o) \tag{7.5}$$

if u is the first child, otherwise:

$$\bar{\mathbf{o}}_u = f(\mathbf{W}_{hi}\mathbf{i}_h + \mathbf{W}_{ho}\mathbf{o}_h + \frac{1}{|\bar{S}(u)|}\sum_{v\in\bar{S}(u)} \mathbf{W}_{dr(v)}\mathbf{i}_v + b_o) \tag{7.6}$$

where $\bar{S}(u)$ is the set of siblings generated before u. $\mathbf{W}_{dr(v)}$ is a weight matrix specific for the dependency relation type $dr(v)$ between v and h. Finally, the outer representation of child u of h is computed as:

$$\mathbf{o}_u = f(\mathbf{W}_{hi}\mathbf{i}_h + \mathbf{W}_{ho}\mathbf{o}_h + b_o) \tag{7.7}$$

if u is the only child, otherwise:

$$\mathbf{o}_u = f(\mathbf{W}_{hi}\mathbf{i}_h + \mathbf{W}_{ho}\mathbf{o}_h + \frac{1}{|S(u)|}\sum_{v\in S(u)} \mathbf{W}_{dr(v)}\mathbf{i}_v + b_o) \tag{7.8}$$

where $S(u)$ is the set of u's siblings. An example of applying IORNN to generate dependency nodes is illustrated in figure 7.2.

Training The IORNN is trained to maximize the probability of generating each word given its partial outer representation:

$$\mathcal{L}(\Theta) = \frac{1}{m}\sum_{T\in D}\sum_{w\in T} \log P(w|\bar{\mathbf{o}}_w) \tag{7.9}$$

where D is the set of dependency trees, and m is the total number of words.

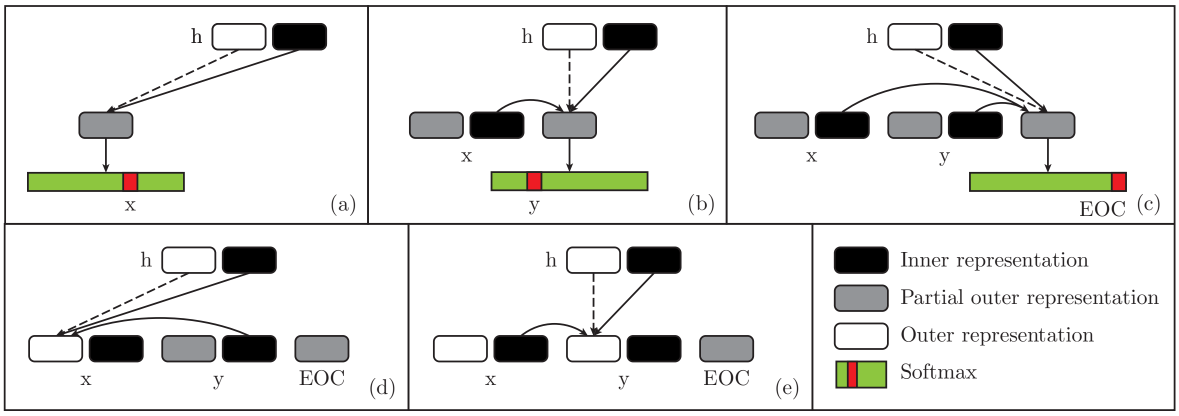


Figure 7.2: An example of applying IORNNs to dependency node h with two children x and y (reproduced from Le and Zuidema (2014)). (a): computing the partial outer representation of x and generating it. (b): computing the partial outer representation of y and generating it. (c): computing the outer partial representation of special node EOC (end of child) and generating it. (d): computing the outer representation of x. (e): computing the outer representation of y.

7.2.2 Discriminative Models

In contrast to generative models, a discriminative reranker learns to distinguish the correct parse tree of a sentence from incorrect ones. Since the tree space is huge, one cannot generate all possible trees to train the model, but can only use a subset of trees generated by a base parser. Therefore, a discriminative reranker is only optimized for a specific parser, and can easily overfit to the error types of the k-best list. In addition, when using the base parser to generate top parses from the train/dev/test sets, there is undoubtedly a huge gap between the quality of those from the training data set (because it is the same data that the base parser was trained on), and those from the development/test data sets, which could reduce the effectiveness of the model. Of course, techniques like n-way jackknifing can be used to eliminate the bias of the training data (Zhou et al., 2016), but they are too costly to apply in practice.

The common idea of all models in this section is to encode the structure of a dependency tree via its node and/or edge representation. Node representation is computed either recursively bottom-up (**RCNN**) or in a step-by-step recurrent manner (**GCN**).

RCNN A RCNN *recursively* encodes each subtree with regards to its children using a *convolutional* layer (see figure 7.3). We denote the input representation of word w_k as $\mathbf{x}_k$, and the phrase (the subtree rooted at w_k) representation output by the RCNN as $\mathbf{h}_k$. In case w_k is a leaf node, $\mathbf{h}_k = \mathbf{x}_k$. Zhu et al. (2015) use word embeddings as

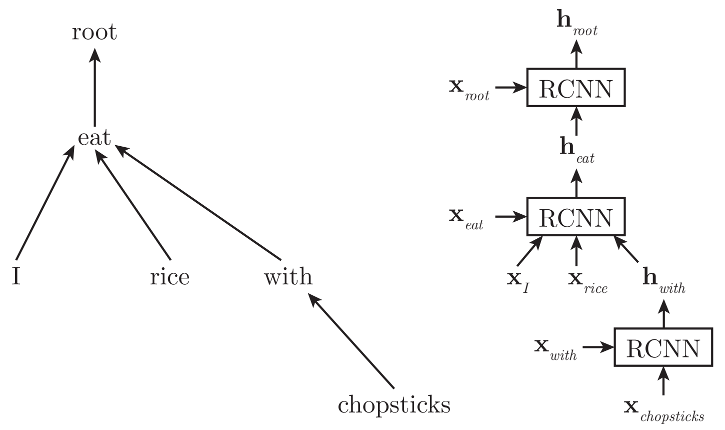


Figure 7.3: An example of applying RCNNs to dependency trees: a sentence and its corresponding computation with RCNNs

input representations.

Figure 7.4 illustrates the architect of an RCNN unit. Given a dependency node h, c_i is its i-th children, $i = 1...K$. Each head-child pair (h, c_i) representation is computed using a convolution layer:

$$\mathbf{z}_i = \tanh(\mathbf{W}^{(h,c)}\mathbf{p}_i) \quad i = 1...K \tag{7.10}$$

where $\mathbf{W}^{(h,c)} \in \mathbb{R}^{m\times n}$ is a weight matrix depending on the POS tag of h and c_i. $\mathbf{p}_i \in \mathbb{R}^n$ is the concatenation of the head input representation $\mathbf{x}_h$, the child phrase representation $\mathbf{h}_{c_i}$, and the distance embeddings $\mathbf{d}^{(h,c_i)}$ of h and c_i:

$$p_i = \mathbf{x}_h \oplus \mathbf{h}_{c_i} \oplus \mathbf{d}^{(h,c_i)} \tag{7.11}$$

The phrase (subtree) representation of h is computed by a max pooling operation on the row of the head-child pair representation matrix $\mathbf{Z}^{(h)}$:

$$\mathbf{Z}^{(h)} = [\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_K] \tag{7.12}$$

$$\mathbf{h}_h[j] = \max_i \mathbf{Z}^{(h)}[i,j] \quad j = 1...m \tag{7.13}$$

The score of the subtree with head h is calculated as:

$$s(h) = \sum_{i=1}^{K} \mathbf{v}^{(h,c_i)} \cdot \mathbf{z}_i \tag{7.14}$$

where $\mathbf{v}^{(h,c)} \in \mathbb{R}^{m\times 1}$ is a weight vector that also depends on the POS tag of h and c_i.

Given a sentence x and its dependency tree y, the score of y is computed by summing the score of all inner nodes h:

$$s(x, y, \Theta) = \sum_{h\in y} s(h) \tag{7.15}$$

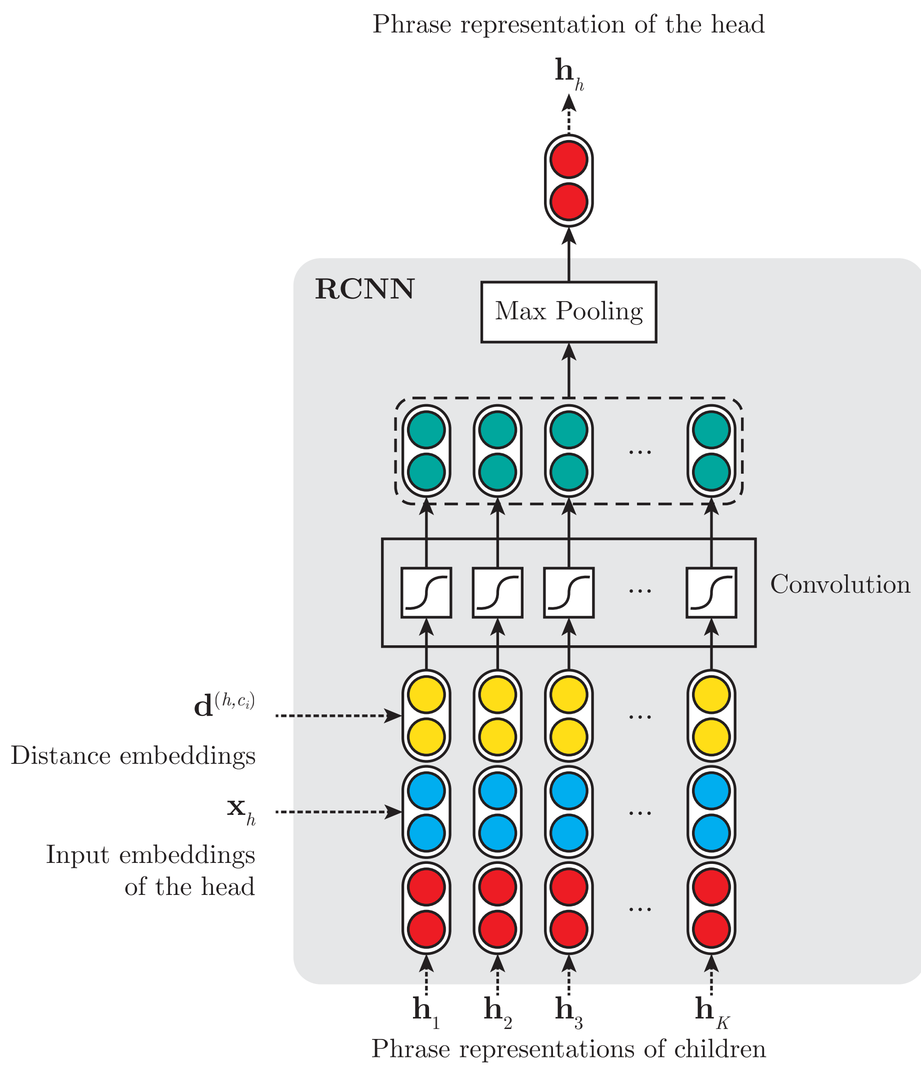

Phrase representation of the head
$\mathbf{h}_h$
RCNN
Max Pooling
Convolution
$\mathbf{d}^{(h,c_i)}$
Distance embeddings
$\mathbf{x}_h$
Input embeddings of the head
$\mathbf{h}_1$ $\mathbf{h}_2$ $\mathbf{h}_3$ $\mathbf{h}_K$
Phrase representations of children

Figure 7.4: The architect of a RCNN unit

The network outputs the predicted tree $\hat{y}$ from the input list $\mathbf{gen}(x)$ with the highest score:

$$\hat{y} = \text{argmax}_{y \in \mathbf{gen}(x)} s(x, y, \Theta) \tag{7.16}$$

The bottom-up fashion used in RCNNs can cause disproportion between the tree structure and its representation due to the order in recursive computation. Consider two trees that only differ in one edge. Their node representations will be more similar if the edge appears higher up in the tree and more different if the edge is close to the lower level since the difference spreads to the upper level. Thus, we believe that a discriminative reranker can benefit from a model that considers nodes in a tree more equally, as in GCNs.

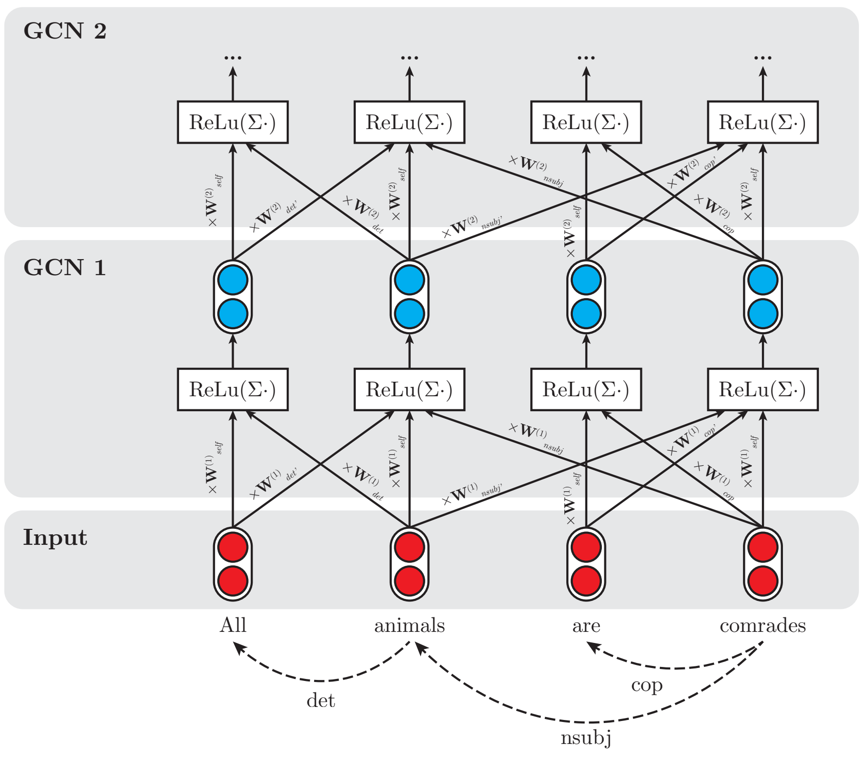


Figure 7.5: An example of syntactic gated GCNs. The bias terms and gates are omitted for simplification. A syntactic function name with an apostrophe (e.g., *subj'*) denotes the edge with the opposite direction to the dependency arc (i.e., from the dependency to the head).

GCN GCNs are used to encode nodes in a graph with information from their neighbors. By stacking several layers of GCNs, the learned representation can capture information about directly connected nodes (with only one layer), or nodes with K hops away (with K layers). We adapt the syntactic gated GCNs for semantic role labeling (figure 7.5) from Marcheggiani and Titov (2017) to encode parse trees in our experiments. To our best knowledge, this is the first time GCNs are used to rank dependency trees.

Let $\mathbf{h}_v^k$ be the representation of node v at step k. A GCN layer computes the representation at the next step as:

$$\mathbf{h}_v^{(k)} = ReLU\left(\sum_{u \in N(v)} g_{u,v}^{(k)}\left(\mathbf{W}_{dir(u,v)}^{(k)}\mathbf{h}_u^{(k-1)} + \mathbf{b}_{L(u,v)}^{(k)}\right)\right) \tag{7.17}$$

where $dir(u, v)$ is the direction of edge (u, v): whether it is directed (1) along, (2) in the opposite direction to the syntactic arc, or (3) is a self-loop; $L(u, v)$ is the syntactic label of edge (u, v). A self-loop is an artificial connection added for the GCN rather than a real loop in the dependency tree (most dependency parsing algorithms cannot produce self-loops or they can be removed in the post-processing step). Its function is to retain the previous information of the node itself in each recurrent step. A reversed edge (opposite direction to the syntactic arc) is also added to make information flows in both ways. The scalar gate $g_{u,v}^{(k)}$ decides the importance of edge (u, v) to the representation of node v:

$$g_{u,v}^{(k)} = \sigma\left(\mathbf{h}_u^{(k-1)} \cdot \hat{\mathbf{v}}_{dir(u,v)}^{(k)} + \hat{\mathbf{b}}_{L(u,v)}^{(k)}\right) \tag{7.18}$$

The input to the GCNs can be word embeddings: $\mathbf{h}_v^{(0)} = \mathbf{x}_{w_v}$, or the hidden representation from the last layer of stacked LSTMs.

The plausibility score of each tree is the sum of the scores of all nodes in the tree:

$$s(x, y, \Theta) = \sum_{v \in y} \mathbf{v} \cdot \mathbf{h}_v^{(K)} \tag{7.19}$$

Training Given an input sentence x, the input to the reranker is the corresponding correct parse tree y and a list of trees generated by a base parser $\mathbf{gen}(x)$. As in conventional ranking systems, all discriminative rerankers can be trained with a margin-based hinge loss so that the score of the correct tree is higher than the score of the incorrect one with a margin of at least m:

$$L(y, t) = \max(0, s(x, t, \Theta) + m - s(x, y, \Theta)) \quad t \in \mathbf{gen}(x) \setminus \{y\} \tag{7.20}$$

Zhu et al. (2015) use a *structured margin* $m = \kappa\Delta(y, t)$, which is computed by counting the number of incorrect edges of t with respect to y:

$$\Delta(y, t) = |e : e \in t, e \notin y| \tag{7.21}$$

κ is a discount hyperparameter indicating the importance of Δ to the loss. In addition, the tree predicted by the model $\hat{y}$ (i.e., the highest scored tree) (7.16) is used to calculate the final loss:

$$\mathcal{L}(\Theta) = \frac{1}{|\mathcal{D}|} \sum_{x,y \in \mathcal{D}} L(y, \hat{y}) \tag{7.22}$$

$$= \frac{1}{|\mathcal{D}|} \sum_{x,y \in \mathcal{D}} \max(0, \max_{t \in \mathbf{gen}(x)} (s(x, t, \Theta) + \kappa\Delta(y, t)) - s(x, y, \Theta)) \tag{7.23}$$

Alternatively, the loss of the predicted tree can be replaced by the average loss of all trees in the list:

$$\mathcal{L}(\Theta) = \frac{1}{|\mathcal{D}|} \sum_{x,y \in \mathcal{D}} \frac{1}{|\mathbf{gen}(x)|} \sum_{t \in \mathbf{gen}(x)} L(y, t) \tag{7.24}$$

$$= \frac{1}{|\mathcal{D}|} \sum_{x,y \in \mathcal{D}} \frac{1}{|\mathbf{gen}(x)|} \sum_{t \in \mathbf{gen}(x)} \max(0, s(x, t, \Theta) + \kappa\Delta(y, t) - s(x, y, \Theta)) \tag{7.25}$$

7.2.3 Mixture Reranking Model

None of the models above does consider the scores from the base parser when ranking trees. In theory, we expect each component to be an expert at different aspects:

- A base parser is optimized to return the highest scored tree (graph-based parser) or transition sequence (transition-based parser).
- A generative reranker is trained to score a node based on the shape of the tree.
- A discriminative reranker learns to recognize mistakes of a base parser.

Therefore, it seems plausible to try combining the advantages from both models, the base parser and the reranker, to produce a better final model. The most common way to do so is to consider the base parser and the reranker as a mixture model.

Mixture reranker The score of any reranking model s_r can be combined with the score of the based parser s_b using a linear combination:

$$s(x, y) = \alpha s_r(x, y, \Theta) + (1 - \alpha)s_b(x, y) \tag{7.26}$$

where $\alpha \in [0, 1]$ is a parameter.

7.3 Evaluating Neural Rerankers for Dependency Parsing

We are now providing a systematic evaluation of different neural reranking models used to rank the k-best lists generated by different parsers. In our first experiments, we try to reproduce the results for the available rerankers (IORNN, RCNN) on English. After that, we compare the performance of the rerankers on German and Czech data. Unless stated otherwise, results are compared based on UAS and LAS *including punctuation*.

7.3.1 Data

English Following Zhu et al. (2015), we use the Penn Treebank (PTB) with standard splits: sections 2-21 for training, section 22 for development, and section 23 for testing. Their reranking models are applied to *unlabeled* trees. The authors used the linear incremental parser from Huang and Sagae (2010) to produce k-best lists and achieved slight improvements due to differences in optimization. In contrast, we obtained the data and pre-trained model Huang and Sagae (2010) from the public repository[1]. Although not emphasized in their paper, Zhu et al. (2015) obtained the top k parses from the *forests* (a by-product of dynamic programming) rather than by using beam search. This is very important for reranking because the forest encodes exponentially many trees and so the k-best list extracted from the parse forest has a higher upper bound (Huang and Sagae, 2010).

Following previous works, we refer to the greedy, one-best results from the base parser as the *baseline*. *Oracle worst* and *best* are the lower and upper bound accuracies of the trees in the k-best list, respectively. *Top tree* results are calculated on the highest scored trees by the base parser in the list.

Table 7.1 shows that both our baseline and upper bound results are lower than those from Zhu et al. (2015). Extracting the top trees from the parse forest results in a much higher upper bound (+3.97%, development set) compared to using beam search (+1.46%). The maximum gain of our k-best list at $k = 64$ using the forest is about 1% lower than in Zhu et al. (2015). Increasing k from 10 to 64 improves the oracle best by 0.75%, but the oracle worst drops by 9%.

German We use the German data set from the SPMRL 2014 Shared Task (Seddah et al., 2014) which contains 50,000 sentences of newspaper text. We follow the original train/dev/test split and use the predicted POS and morphological tags provided by the shared task organizers. The top k parses are produced using the graph-based parser in the MATE tools (Bohnet, 2010)[2], a non-neural model that employs second order, approximate non-projective parsing (McDonald and Pereira, 2006). The algorithm first finds the highest scored projective tree with exact inference, then rearranges the edges one at a time as long as the overall score improves and the parse tree does not violate the tree constraint. This algorithm also creates a list of k-best trees through its search process. We also tried to generate the k-best lists with a transition-based parser by adding a beam search decoder, but the beam failed to improve the parsing upper bound.

Czech We use the Czech Universal Dependencies (UD) Treebank[3], based on the Prague Dependency Treebank 3.0 (Bejček et al., 2013). We use the original train/dev/test split and use MarMoT (Mueller et al., 2013) to predict UD POS tags by 5-way jackknifing. The k-best lists are created using the same parser as for German.

The properties of the k-best lists extracted from the German and Czech data

[1]https://github.com/lianghuang3/lineardpparser
[2]https://code.google.com/p/mate-tools
[3]https://universaldependencies.org/

	UAS w/ punct.		UAS w/o punct.	
Data Set	Dev	Test	Dev	Test
Zhu et al. (2015)				
Baseline	_	_	92.45	92.35
$k = 64$				
Oracle worst	_	_	73.30	_
Oracle best	_	_	97.34	_
Huang and Sagae (2010)				
Baseline	91.34	91.45	92.09	92.05
$k = 10$, forest				
Top tree	91.34	91.45	92.09	92.05
Oracle worst	79.68	79.56	80.21	80.19
Oracle best	95.31	95.33	95.99	95.82
$k = 64$, forest				
Top tree	91.34	91.45	92.09	92.05
Oracle worst	70.62	70.72	71.26	71.51
Oracle best	96.06	96.15	96.65	96.55
$k = 64$, beam				
Top tree	91.28	91.30	92.05	91.89
Oracle worst	68.16	68.19	71.97	71.95
Oracle best	92.80	92.93	93.81	93.76

Table 7.1: k-best list accuracy from PTB. Top: accuracies reported in Zhu et al. (2015). Bottom: extracted lists from Huang and Sagae (2010)'s model, either using the parse forest or beam search.

are shown in table 7.2. Extracting the top k parses results in scores lower than the baseline when using the top trees as output, as the reranking scores do not always correlate with the quality of the trees. Besides the top 50 tree list, we use a shorter 10-best list ($k = 50$, top 10) by selecting only the top results rather than re-computing the list using a smaller beam size to save computation cost.

Pre-trained word embeddings In all experiments on English, we use the 50-dimensional GloVe word embeddings (Pennington et al., 2014) trained on Wikipedia 2014 and Gigaword 5. For German, we train 100-dimensional dependency-based word embeddings (Levy and Goldberg, 2014) on the SdeWaC corpus (Faaß and Eckart, 2013) with a cutoff frequency of 20 for both words and contexts and set the number

	Dev		Test	
Data Set	UAS	LAS	UAS	LAS
German				
Baseline	92.91	91.04	90.19	87.90
$k = 50$				
Top tree	91.75	90.04	88.36	86.28
Oracle worst	81.20	79.48	79.04	77.12
Oracle best	96.40	95.08	93.51	91.71
$k = 50$, top 10				
Top tree	91.75	90.04	88.36	86.28
Oracle worst	84.42	82.56	82.09	79.96
Oracle best	95.56	94.10	92.54	90.64
Czech				
Baseline	92.22	89.30	91.87	88.85
$k = 50$				
Top tree	91.02	88.28	90.74	87.93
Oracle worst	82.24	79.68	81.98	79.32
Oracle best	95.04	92.71	94.70	92.29
$k = 50$, top 10				
Top tree	91.02	88.28	90.74	87.93
Oracle worst	85.20	82.42	84.95	82.08
Oracle best	94.19	91.72	93.82	91.24

Table 7.2: k-best list accuracies for the German SPMRL and Czech UD data sets.

of negative samples to 15. In experiments on Czech, we reduce the number of dimensions of the word vectors from fastText (Bojanowski et al., 2017) to 100 using PCA (Raunak et al., 2019).

7.3.2 Reproducing Reranking Results for PTB

This section is dedicated to the reproduction of the published results for the **IORNN** and **RCNN** rerankers on the English PTB. All results are from one run since we observe little variation between different runs[4] (and even between different settings the results hardly vary).

[4]For instance, the standard deviations of 5 runs on the development and test sets are $\sigma_{dev} = 0.05$, $\sigma_{test} = 0.07$ (%) when running the best GCN model setting on the English data.

Model	UAS	LAS
Le and Zuidema (2014)		
Baseline	91.99	89.97
Oracle best ($k = 10$)	96.24	93.73
Reranker ($k = 6$)	92.83	90.76
Mixture ($k = 9$)	93.08	91.02
Reproduction on Le and Zuidema (2014)'s data		
Baseline	92.19	90.11
Oracle best ($k = 10$)	96.25	93.67
Reranker ($k = 10$)	92.58	90.54
Mixture ($k = 10$)	93.01	90.91
Reproduction on our data		
Baseline	91.45	_
Oracle best ($k = 10$)	95.33	_
Reranker ($k = 10$)	91.70	_
Mixture ($k = 10$)	92.06	_

Table 7.3: The IORNN reranker performance on the PTB test set

IORNN The results from Le and Zuidema (2014) can be reproduced with 93.01% UAS using the data and instructions from the public repository[5]. We are able to replicate this trend on our unlabeled English data described in section 7.3.1, i.e., the reranking results are better than the baseline. The IORNN mixture model achieves 92.06% UAS on the test set, which is lower than the reproduced results on the paper's original data. Our baseline, however, is also lower due to the use of different data conversion rules for the conversion from constituency trees to dependencies, and the use of different base parsers. On our unlabeled English data described in section 7.3.1, the IORNN mixture model achieves 92.06% UAS on the test set, which is lower than the reproduction result but our baseline is also lower. Note that Le and Zuidema (2014) also optimize the results on k, while we keep k fixed in experiments with our data to make the results comparable to that of other models. In addition, the authors do a logarithmic scaling for the score of the reranker in the mixture model combination (equation 7.26):

$$s(x, y) = \alpha \log(s_r(x, y, \Theta)) + (1 - \alpha) s_b(x, y) \tag{7.27}$$

[5] https://github.com/lephong/iornn-depparse

Models	Dev UAS	Test UAS
Baseline	91.34	91.45
RCNN	90.60	90.04
(1) -L2	90.54	90.28
(2) -largest margin (average margin)	90.51	**90.29**
(3) -margin discount ($\kappa = 1.0$)	90.48	90.05
(4) -structured margin	90.50	89.99
(1) + (2) + (3) + (4)	**90.65**	**90.29**

Table 7.4: Ablation study on the RCNN reranker (PTB, $k = 10$)

and we use this function as it is.[6,7]

Table 7.3 summarizes the results from our reproduction study.

RCNN Since the codes are not publicly available, we re-implement the RCNN model following the description in the paper (Zhu et al., 2015). However, we were not able to reproduce the results on the 10-best list extracted from the parse forest. The authors report 93.83% (+1.48) UAS without punctuation using the mixture reranker with $k = 64$, and the same trend sets for all k. All our attempts to get better results than the base parser fail. Even when combining the reranking score with the score from the base parser, results do not improve over the baseline.

We run an ablation study to investigate the effect of different hyperparameters on the model's performance (see table 7.4). Those includes:

1. Removing L2 regularization term
2. Replacing the largest margin (equation 7.22) with the average margin (equation 7.24)
3. Reducing the margin discount κ from 2.0 to 1.0
4. Replacing the structured margin with a margin of 1.0

The results suggest that removing L2 and replacing the largest margin with the average margin decrease the accuracies on the dev set, but increase those on the test set. The margin discount parameter κ and the structure margin unfortunately did not show much effect. Finally, we achieve the best scores (UAS 90.65% and 90.29%)

[6]The IORNN code does not output the reranking scores to train a mixture model separately.

[7]Applying a scaling to either score only affects the range of the combination parameter α, not the final results.

Models	Dev UAS	Test UAS
Baseline	91.34	91.45
RCNN	90.65	90.29
RCNN-shared		
structure margin, $\kappa = 2.0$	90.96	90.62
structure margin, $\kappa = 1.0$	90.97	90.63
-structure margin, $\kappa = 1.0$	90.94	90.78
RCNN-shared (+BiLSTMs)		
structure margin, $\kappa = 2.0$	91.47	91.34
structure margin, $\kappa = 1.0$	91.44	91.42
-structure margin, $\kappa = 1.0$	**91.50**	**91.46**

Table 7.5: Ablation study on the RCNN-shared reranker (PTB, $k = 10$). All models are trained with the average margin loss, without L2 regularization. +BiLSMTs models have 2 additional LSTMs layers.

on both development and test sets when removing L2 and structured margin and replacing the largest margin with the average margin. However, one thing we noted during training is that the learning curves indicate severe overfitting. In conclusion, despite our efforts, we were not able to reproduce the RCNN results from Zhu et al. (2015).

RCNN-shared As the learning curves for the RCNN models show severe overfitting, we propose to simplify the original model. The original RCNN has a large number of parameters because it uses different weight matrices and vectors for the POS tags of the current head-child pair. In the simplified model, we replace those matrices $\mathbf{W}^{(h,c)}$ and vectors $\mathbf{v}^{(h,c)}$ in equations 7.10 and 7.14 with a shared matrix $\mathbf{W}$ and vector $\mathbf{v}$. Word embeddings and POS embeddings (randomly initialized) are concatenated as the input to the RCNN. Following common practice, we also test a model where we place several BiLSTM layers before the RCNNs to learn better representations from the input embeddings (+BiLSTMs).

We report the performance of our models on the PTB with top parses $k = 10$ (table 7.5). By switching from RCNN to the RCNN-shared model, we are now able to beat the baseline, even though by only a small margin (UAS 90.50% and 91.46% on the dev and test sets respectively). A new ablation study confirms previous results for the RCNN model: results are better without using L2, margin discount, and structured margin. The largest margin loss does not work with the RCNN-shared

k_{train}	k_{eval}	UAS
10	10	91.50
64	10	91.86
10	64	90.82
64	64	91.62

Table 7.6: Accuracies of the RCNN-shared (+BiLSTMs) model on the PTB development set with regard to the size of the k-best list

n_{LSTMs}	n_{GCNs}	Dev UAS	Test UAS
1	2	**92.44**	92.17
1	3	92.40	**92.23**
2	1	92.27	92.01
2	2	92.32	**92.23**
2	3	87.03	87.00

Table 7.7: Accuracies of the GCN reranker on the PTB development set. All models are trained on $k = 64$ and evaluated on $k = 10$.

models as the UAS on the development set (86.87%) stops improving before the end of the first training epoch.

We also study the effect of k on the model's performance (table 7.6). Training the reranker on a larger k-best list[8] improves the UAS by 0.36% on the development set, which shows that the model learns better with more negative examples. Increasing k at test time, on the other hand, hurts performance because the longer list now contains more low-quality trees (see table 7.1, the oracle best increases by 0.75% while the oracle worst decreases by 9.06% on the development set when increasing k from 10 to 64). The drop caused by using a longer list at test time is also smaller (0.20% vs. 0.68%) when the model is trained with more trees.

7.3.3 Reranking with GCNs

We now present results for our new GCN reranking model on the English data. The best GCN model (using 1 BiLSTM layer and 3 GCN layers) trained on $k = 64$

[8]In practice, we do not train on the whole k-best trees when k is large, but down-sample k in each batch to keep the training time efficient. See appendix A.7.1 for details.

Model	UAS
Baseline	91.45
IORNN ($k_{train} = 10$)	
Reranker ($k_{test} = 10$)	91.70
Mixture ($\alpha = 0.015$)	92.06
RCNN ($k_{train} = 10$)	
Reranker ($k_{test} = 10$)	90.29
Mixture ($\alpha = 0.005$)	91.53
With oracle	92.34
RCNN-shared (+BiLSTMs) ($k_{train} = 64$)	
Reranker ($k_{test} = 10$)	91.75
Mixture ($\alpha = 0.05$)	91.92
With oracle	94.37
Reranker ($k_{test} = 64$)	91.43
Mixture ($\alpha = 0.01$)	92.21
With oracle	93.56
GCN ($k_{train} = 64$)	
Reranker ($k_{test} = 10$)	92.23
Mixture ($\alpha = 1.0$)	92.23
With oracle	95.25
Reranker ($k_{test} = 64$)	92.11
Mixture ($\alpha = 0.01$)	**92.48**
With oracle	94.69

Table 7.8: Results for different rerankers (PTB test set)

parse trees significantly outperforms the RCNN-shared model[9] with 92.40% UAS on the development set, compared to 91.86% for RCNN-shared ($p < .001$), an increase of +0.54%. From table 7.7, we see that there is little difference for increasing the number of GCN layers from 2 to 3. The GCN model stops learning well when we increase the number of LSTM layers to 2 and the number of GCN layers to 3 due to over-parameterization.

The best results for different reranking models on the PTB test set are summarized

[9]We did not do a hyperparameter optimization, but increased the number of parameters in the best RCNN-shared models and observed no significant improvement. The hyperparameters for different models in our experiments are listed in the appendix.

Model	UAS w/o punct.	Δ
Zhu et al., 2015		
Baseline	92.35	
Mixture reranker	93.83	+1.48
With oracle	94.16	
Ours (Mixture GCNs)		
Baseline	92.05	
Mixture reranker	93.06	+1.01
With oracle	94.99	

Table 7.9: Reranking reproduction results on the PTB test set ($k = 64$)

in table 7.8. *Reranker* is the ranked list produced by the reranking model only. *Mixture* is the result for combining the output score given by the rerankers and the score of the base parser as described in section 7.2.3. Following Zhu et al. (2015), we do not use the exact linear equation (7.26), but do logarithmic scaling of the base parser's score:

$$s(x, y) = \alpha s_r(x, y, \Theta) + (1 - \alpha) \log(s_b(x, y)) \tag{7.28}$$

The parameter α is optimized based on the results on the development set, which has the same k as the test set. Since the correct tree is not always in the k-best list, we show an upper bound performance of our rerankers where we manually add the gold trees to the input list (*with oracle*). Note that *with oracle* is the result from the reranker, not from the mixture reranking model because the correct tree does not have a score from the base parser if it is not included in the k-best list.

Combining the score from both the reranker and the base parser consistently improves over the reranking score alone (except for the GCN reranker $k_{test} = 10$), which confirms our hypothesis that the parser and the reranker complement each other by looking at different scoring criteria. Although the accuracy drops when reranking longer lists, the mixture scores are always higher. Compared to the RCNN-shared models, the GCN models benefit less from the mixture models, maybe because the GCNs rank trees more similar to the base parser.

The upper bound performance (*with oracle*) shows that we can still improve results with a better k-best list. Interestingly, although we achieve modest improvement compared to Zhu et al. (2015), our upper bound is higher than theirs. A comparison of results with the original RCNN paper on their data is given in table 7.9.

We further report the top 10 UAS improvements on frequent ($n \geq 500$) POS tags

POS	Total	Baseline	GCNs	Δ
RB	1,939	83.75	86.59	2.84
VBG	877	85.29	87.34	2.05
”	511	88.26	90.22	1.96
“	531	79.47	81.17	1.70
VB	1,571	93.06	94.53	1.47
TO	1,240	93.23	94.68	1.45
VBP	791	91.53	92.92	1.39
MD	584	93.15	94.52	1.37
,	3,062	81.55	82.85	1.30
CC	1,367	87.56	88.81	1.25

Table 7.10: UAS of the GCN mixture model on the PTB test set ($k = 64$) based on the (selected) POS tag of the modifier words. The complete results are in table B.1.

of the modifier words in table 7.10. Except for a minor reduction in 5 categories (see the complete table B.1), our best model (mixture reranker with GCNs) outperforms the baseline for all other POS tags. Head attachment improves the most in adverbs (RB), verbs (VB, VBG, VBP), prepositions *to* (TO), and conjunctions (CC).

7.3.4 Neural Reranking for MRLs

We now evaluate the reranking models that have proved to be effective for English (IORNN, RCNN-shared (+BiLSTMs), and GCNs) on German and Czech data. Note that the RCNN model only ranks *unlabeled* trees while the other two models also consider the dependency labels, which is particularly important for non-configurational languages. All models are trained with the same hyperparameters settings as for English. The mixture scores are combined using equation 7.26 except that we optimize the IORNN mixture model using the original tool provided by the authors.

The results from different reranking models are presented in tables 7.11 and 7.12. Neither the IORNN nor the RCNN-shared rerankers can surpass the baseline. After combining their scores with the score of the base parser (*mixture*), the RCNN-shared mixture model gets a similar accuracy to the base parser, while the IORNN is still 0.5% behind. The GCN mixture model is the only model that gets significant improvements over all other models ($p < .001$) including the baseline, although small (∼0.5% LAS).

The base parser's top tree results are about 1.5% below the baseline (table 7.2), showing that the ranking produced by the base parser is not very good. Therefore,

Model	UAS	LAS
Baseline	90.19	87.90
Top tree	88.36	86.28
IORNN ($k_{train} = 10$)		
Reranker ($k_{test} = 10$)	89.32	87.16
Mixture ($\alpha = 0.91$)	89.47	87.41
RCNN-shared ($k_{train} = 50$)		
Reranker ($k_{test} = 10$)	89.94	87.35
Mixture ($\alpha = 0.1$)	90.16	88.01
With oracle	93.97	92.14
Reranker ($k_{test} = 50$)	89.47	86.30
Mixture ($\alpha = 0.1$)	90.15	87.93
With oracle	92.91	90.42
GCN ($k_{train} = 50$)		
Reranker ($k_{test} = 10$)	90.22	88.16
Mixture ($\alpha = 0.1$)	90.37	**88.44**
With oracle	95.37	94.21
Reranker ($k_{test} = 50$)	90.01	87.74
Mixture ($\alpha = 0.09$)	**90.41**	88.43
With oracle	94.43	92.90

Table 7.11: Performance of different rerankers on the German SPMRL test set

Model	UAS	LAS
Baseline	91.87	88.85
Top tree	91.02	88.28
IORNN ($k_{train} = 10$)		
Reranker ($k_{test} = 10$)	91.07	87.97
Mixture ($\alpha = 0.94$)	91.42	88.54
RCNN-shared ($k_{train} = 50$)		
Reranker ($k_{test} = 10$)	91.29	87.59
Mixture ($\alpha = 0.07$)	91.77	88.78
With oracle	94.51	91.91
Reranker ($k_{test} = 50$)	90.68	86.63
Mixture ($\alpha = 0.07$)	91.79	88.80
With oracle	93.28	89.99
GCN ($k_{train} = 50$)		
Reranker ($k_{test} = 10$)	91.55	88.43
Mixture ($\alpha = 0.11$)	91.87	88.98
With oracle	95.59	93.97
Reranker ($k_{test} = 50$)	91.12	87.84
Mixture ($\alpha = 0.09$)	**91.89**	**89.01**
With oracle	94.47	92.42

Table 7.12: Performance of different rerankers on the Czech UD test set.

	German		Czech	
Label	Baseline	Δ_{GCNs}	Baseline	Δ_{GCNs}
nsubj	89.20	1.48	91.50	0.46
obj	82.84	1.91	90.10	0.54
iobj	67.25	1.15	60.92	2.22
conj	81.78	0.72	74.15	1.23

Table 7.13: Labeled F1 differences between the baseline and the GCN mixture model for selected dependency types from the German and Czech test sets. The complete results are in tables B.2 and B.3 in the appendices.

combining the scores in the mixture models does not yield substantial improvements.

Taking a closer look at different grammatical functions in the output, we can see a clear difference between the reranking results and the baseline (table 7.13). Although the overall accuracy is similar, our reranking results show a better performance for core arguments (nsubj: subject, obj: direct object, iobj: indirect object) and conjunctions (conj).

7.4 Analysis

Through our experiments, we have shown that neural reranking models, which have demonstrated their effectiveness on English data, fail to improve baseline parsing results when applied to German and Czech. This brings us to the question of whether this failure is due to the differences between the languages or simply due to the lower quality in the German and Czech k-best lists that are input to the rerankers. It is conceivable that language-specific properties such as the freer word order and richer morphology in German and Czech might make it harder for our models to learn a good representation capturing the quality of a specific parse tree. However, when we add the correct parse tree to the k-best list (*with oracle* results in tables 7.8, 7.11, and 7.12), the accuracy goes up to 94% for English, German and Czech, which effectively eliminates the first reason.

This points to the method used to obtain the k-best list as the main factor responsible for the low results for German and Czech. Beam search, although being straightforward to implement, fails to create high-quality k-best lists for the base parsers used for both languages (section 7.3.1). While several projective parsers support k-best parsing (Huang and Sagae, 2010; McDonald and Pereira, 2006), there is, to the best of our knowledge, no out-of-the-box parsing system that implements

	English		German			Czech		
Length	Total	UAcc	Total	UAcc	LAcc	Total	UAcc	LAcc
10	225	97.33	1,175	91.74	84.94	2,750	94.11	85.27
20	725	85.10	1,820	78.46	65.22	3,751	80.81	63.77
30	795	67.30	1,271	50.83	38.79	2,402	58.49	37.84
40	465	56.99	484	25.21	16.12	863	33.26	18.77
50	162	35.19	165	10.30	4.24	265	14.34	4.91
>50	44	11.36	85	3.53	1.18	117	3.42	0.85
All	2,416	70.28	5,000	65.86	55.28	10,148	72.46	57.37

Table 7.14: Unlabeled (UAcc) and labeled tree accuracy (LAcc) of the k-best list from the English PTB ($k = 64$), German SPMRL, and Czech UD test sets ($k = 50$) test sets

an effective non-projective k-best parsing algorithm (as, for example, Hall (2007)'s algorithm).

Gold tree ratio Clearly, the (upper bound) tree accuracy of the k-best list decides the reranking performance. In all data sets, we observe that the accuracy decreases when the sentence length increases (table 7.14). Overall, the (unlabeled) tree accuracy in the English k-best list is ~5% higher than in the German data but is behind that in the Czech data. This, however, is not caused by a larger amount of long sentences in the German data. For sentences of the same length, the top k trees from the PTB contain more gold trees than those from the German SPMRL and Czech UD data sets.

We further study the effect of the gold tree ratio for reranking by removing the gold trees from the k-best list to reduce the ratio to a certain level. Figure 7.6 shows that the gold tree ratio strongly correlates with the reranking results.

***k*-best list variation** We measure the variation between the trees in the k-best lists by calculating the standard deviation of their UAS. Figure 7.7 illustrates the UAS standard deviation distribution in the data for the three languages for $k = 10$. In each data set, the tree UAS variation in the English data is the highest, followed by German and then Czech, which shows that the re-arranging method used to generate German and Czech k-best trees tends to return more similar trees. We hypothesize that reranking benefits from diversity, especially if the data contains hard negative examples (incorrect trees that are very similar to the correct one). The gap between reranker performance and *with oracle* results shows that the reranker is able to detect

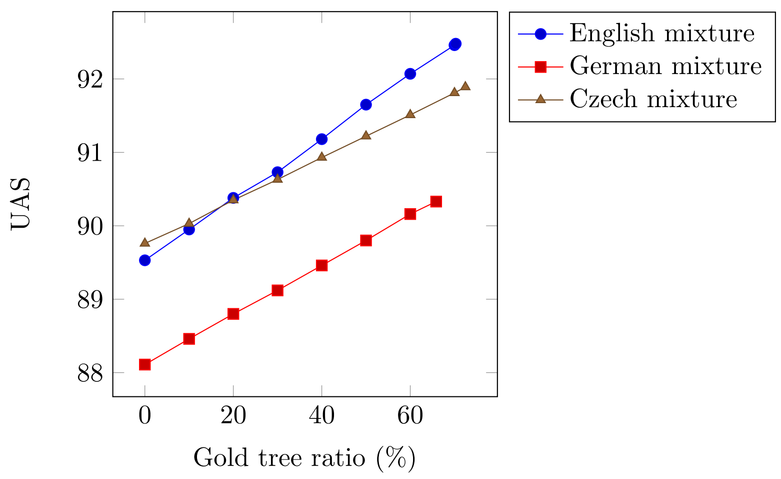


Figure 7.6: UAS of the GCN reranking model with respect to the gold tree ratio in the k-best lists

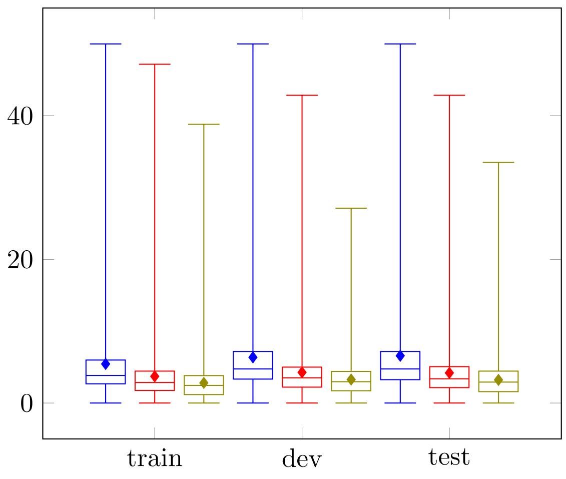


Figure 7.7: Tree UAS standard deviation of 10-best lists. From left to right: English, German, Czech.

the correct tree among the incorrect ones because they are very different from each other.

Reranking models Among the neural rerankers, the RCNNs are prone to error propagation from the lower levels, and the IORNNs are sensitive to the order of the child nodes. Both models did not work very well when moving to German and Czech compared to the GCNs, which disregard the top-down or left-to-right order.

In practice, parser output reranking is not a very cost-effective way to improve parsing performance, unless we have a fast way to generate high-quality output trees. However, the small improvement in core arguments might be useful for downstream applications that require high-quality prediction of core arguments.

7.5 Summary

In this chapter, we have studied reranking as a technique to improve parsing at the sentence level. We have evaluated recent neural techniques for reranking dependency parser output for English, German and Czech and presented a novel reranking model, based on graph convolutional networks (GCNs). We were able to reproduce results for English, using existing rerankers, and showed that our novel GCN-based reranker even outperformed them. However, none of the rerankers works well on the two morphologically rich(er) languages.

Our analysis gave some insights into this issue. We showed that the failure of the rerankers to improve results for German and Czech over the baseline is due to the lower quality of the k-best lists. Here the gold tree ratio in the k-best list plays an important role, as the discriminative rerankers are very well able to distinguish the gold trees from other trees in the list, but their performance drops notably when we remove the gold trees from the list. In addition, we observe a higher diversity in the English k-best list, as compared to German and Czech, which helps the rerankers to learn the differences between high- and low-quality trees.

We conclude that the prerequisite for improving dependency parsing with neural reranking is a diverse k-best list with a high gold-tree ratio. The latter is much harder to achieve for MRLs where the freer word order and high amount of non-projectivity result in a larger number of tree candidates, reflected by a lower gold tree ratio.

CHAPTER 8

Conclusion & Outlook

8.1 The Contributions of this Thesis

In this thesis, we have studied modeled linguistic information on different levels and studied the effect of our models on German dependency parsing. Namely, inspired by the linguistic properties at the level of (sub)words, syntax, semantics, and the sentence level, we proposed new methods based on neural networks to improve dependency parsing performance for German.

In summary, the work presented in this thesis has:

- shown the effect of language-specific properties on dependency parsing for German at each linguistic level;
- demonstrated that assumptions from language-agnostic parsing research do not necessarily apply to German (or other morphologically rich(er) languages);
- developed better techniques for parsing German texts.

We will now discuss each contribution in detail.

8.1.1 The Effect of Modeling on Different Linguistic Levels

We have hypothesized that parsing requires modeling different levels of linguistic structures, and thus have shown the effect of each level on dependency parsing performance.

We started with looking into the first structural level of parsing, i.e., the **(sub)word** level (chapter 4). We addressed the problem of out-of-vocabulary words in German caused by *compounding* and introduced a new type of subword embeddings, the *compound embeddings*, with the hope of providing more information about an unknown compound via the knowledge of its components. Our experiments have shown that compound embeddings only outperform word embeddings in dependency parsing when the part-of-speech (POS) information is *absent*, and that character-based embeddings always performed better than both of them together. Thus, we concluded that it was not the *semantic* information of the unknown compounds that was crucial for parsing German, but their *morpho-syntactic* information. This explains why the effect of compound embeddings in dependency parsing was leveled out when POS information was present, and how the character-based embeddings (that encodes morpho-syntactic information) achieved similar performance as the setting with POS tags.

We then investigated challenges for grammatical function labeling on the **structural and word order** level (chapter 5). *Case syncretism* in German is a major source of ambiguity for local grammatical function labelers that label each edge independently. Although truly ambiguous cases cannot be resolved even by humans, in cases, mistakes can be avoided by considering the whole context when assigning the labels. We have shown that grammatical function labeling can be improved by augmenting the labeler of a neural dependency parser with a *decision history*, using Long Short-Term Memory networks (LSTMs). All our proposed models significantly outperformed the baseline for three languages: English, German, and Czech, but the best improvement on core argument functions was achieved when ordering the tree nodes according to the breadth-first traversal (BFS) from the root. The impact of those history-based labeling models was more prominent in German and Czech than in English, not because the models were better at handling long dependencies but, instead, were better at dealing with the uncertainty in head direction.

We studied the interaction of syntactic parsing with the **semantic** level via the problem of PP attachment disambiguation in chapter 5. We presented a new system for PP attachment disambiguation that employed biaffine attention and utilized pre-trained contextualized word embeddings as semantic knowledge. Our system outperformed previous work on German by a large margin just by considering all words in a sentence as candidate heads, thus avoiding error propagation due to using predicted syntactic information for candidate extraction. Despite that, its performance was still worse than that of a strong neural parser with access to the same semantic knowledge. Our experiments suggested that parsing systems were in general superior to systems specialized for PP attachment disambiguation since neural parsers could make more efficient use of the training data by using a joint classifier for all relation types.

Finally, chapter 7 explored syntactic parsing at the **sentence** level via *reranking*. We presented a novel reranking model, based on graph convolutional networks (GCNs) that outperformed previous approaches on neural reranking for English. However, all neural reranking models, which had demonstrated their effectiveness on English data, failed to improve baseline parsing results when applied to two morphologically rich(er) languages, German and Czech. Our analysis pointed out that the failure was due to the lower quality of the k-best lists, where the gold tree ratio and the diversity of the list played an important role.

8.1.2 The Presumption of Language-Agnostic Approaches

Despite being considered *language-agnostic*, some models are designed with specific linguistic properties in mind, which may cause problems when the target language does not share the same characteristics. Our results, as summarized in section 8.1.1 above, illustrate this.

In particular, we have learned that the *word order* has a profound effect on modeling. First, at the structural level, we have shown that labeling the grammatical function of each edge independently does not work well with languages with a free word order, thus better performance can be achieved by taking into account *history*.

As the second case in point, at the sentence level, we have found that the inside-outside recursive neural network (IORNN) reranker and the recurrent convolutional neural network (RCNN) reranker do not work very well when being applied to German and Czech, despite working well on English data. It is very likely that the IORNN reranker encodes trees with left-to-right order while German and Czech have a freer word-order, and the RCNN reranker computes tree scores in a bottom-up fashion, thus being prone to error propagation when the trees in the k-best list are of low(er) qualities (as in German and Czech). Hence, our proposed reranker with graph convolutional networks (GCNs) that disregards the bottom-up or left-to-right order performs the best among the three models.

8.1.3 Better Parsing Models for German

Our insights into the impact of different types of information on different structural levels have helped us to construct better techniques for parsing German. In this thesis, we have introduced the following novel methods:

- A grammatical function labeler with a decision history (chapter 5). Our labeler with BFS ordering significantly outperforms the baseline labeler in three languages: English, German, and Czech.
- A PP attachment disambiguation system with biaffine attention that takes into account all words in the sentence as candidates (chapter 6). Compared

to previous work, our system avoids error propagation caused by candidate extraction, thus outperforming the reference system by a large margin.

- A neural reranker with GCNs (chapter 7). Our model outperforms previous models on English and is the only model that is able to improve results over the baseline on German and Czech.

8.2 Future Work

With the goal of developing better techniques for parsing German, this section outlines some possible future directions based on the work presented in this thesis.

8.2.1 Improving Dependency Parsing at All Levels

This thesis has looked at different levels of linguistic structures and improved parsing at each level independently. What is still left to explore is a parsing system that takes into account the specific properties at all levels. In future work, we hope to develop and experiment on such a system to understand the combined effect of different structural levels on parsing German.

8.2.2 Unifying Different Treebanks

The PP attachment disambiguation system that we proposed in chapter 6 can utilize data from different treebanks, as long as they agree on the definition of PP head (function vs. content head). This opens a new line of work that attempts to exploit treebanks based on different underlying linguistic theories to improve parsing. This could be achieved by considering different treebanks as different tasks in a multi-task model.

8.2.3 Generating More Trees

We have seen that adding pre-trained language models to an existing system can greatly improve the performance of that system (section 2.2.4). However, such practice considerably increases the number of parameters of the current system, making it bigger and slower. In addition, we have not yet investigated the effect of incorporating pre-trained language models into parsers in a systematic fashion, and whether this can achieve the same effect as adding more training data. This is another area that we hope to explore in the future. Additional training data can be created by data augmentation, automatically processing, or based on Natural Language

Generation (NLG) models. For instance, this might be an idea for creating training data for PP attachment, based on attachment information taken from treebanks and using NLG models to generate new variants for certain attachments as training data.

APPENDIX A

Hyperparameters and Training Details

A.1 Head-Selection Parser

The head-selection parser used in chapters 4 and 5 is our re-implementation of the parser described in Zhang et al. (2017).

Hyperparameter	**Value**
Lexical	
Word cutoff	5
Lowercase	False
Training	
Optimizer	Adam
Learning rate	0.001
Batch size	50
Max epoch	20
Regularization	
Max norm	5.0
Input dropout	0.05
Dropout	0.5
L2	0.001
Neural Network	
Word emb dim	300
Tag emb dim	40
Character-based word embeddings	
Character emb dim	50
Hidden dim	100

Hyperparameter	**Value**
Compound embeddings	
Lexeme emb dim	50
Hidden dim	100

Table A.1: Hyperparameters of the head-selection parser used in chapters 4 and 5

A.2 LSTM Labelers

The LSTM labelers in chapter 5 consist of three different models: BiLSTM(l), BiLSTM(b), and TreeLSTM. They were trained on top of the unlabeled trees returned by the head-selection parser (see appendix A.1).

Hyperparameter	**Value**
Training	
Optimizer	Adam
Learning rate	0.001
Batch size	50
Max epoch	10
Regularization	
Max norm	5.0
Input dropout	0.05
Dropout	0.5
L2	0.001
Neural Network	
LSTM hidden dim	200

Table A.2: Hyperparameters of the LSTM labelers used in chapter 5

A.3 Biaffine Parser

The reference parser used in chapter 6 is our re-implementation of the dependency parser with biaffine attention described in Dozat and Manning (2017).

Hyperparameter	**Value**
Lexical	
Word cutoff	2
Lowercase	True
Training	
Loss	Cross entropy
Optimizer	Adam
1st momentum β_1	0.9
2nd momentum β_2	0.9
Learning rate	0.002
Decay rate	0.75
Decay step	5000
Batch size	5000 (words)
Max epoch	200
No. train buckets	40
No. dev buckets	10
Regularization	
Max norm	5.0
Input dropout	0.2
Layer dropout	0.33
Recurrent dropout	0.33
L2	0
Neural Network	
Word emb dim	100
Tag emb dim	100
LSTM hidden dim	400
Edge projection dim	500
Label projection dim	100
No. LSTMs	3

Table A.3: Hyperparameters of the biaffine parser used in chapter 6

A.4 PP Attachment Disambiguation System: PP-REP

Hyperparameter	**Value**
Lexical	
Word cutoff	3
Lowercase	False
Training	
Loss	Hinge
Optimizer	Adagrad
Learning rate	0.005
Decay rate	0.75
Decay step	5000
Batch size	32
Max epoch	50
Regularization	
Max norm	None
Input dropout	0
Dropout	0
L2	0
Neural Network	
Hidden dim	1000

Table A.4: Hyperparameters of the PP attachment disambiguation system PP-REP (section 6.3.1)

A.5 Topological Field Labeler

Hyperparameter	**Value**
Lexical	
Word cutoff	2
Lowercase	True
Training	
Optimizer	Adam
Learning rate	0.002
Decay rate	0.75
Decay step	5000

Hyperparameter	**Value**
Batch size	512 (words)
Max epoch	10
No. train buckets	40
No. dev buckets	10
Regularization	
Max norm	5.0
Input dropout	0.2
Dropout	0.33
L2	0
Neural Network	
Word emb dim	100
Tag emb dim	50
LSTM hidden dim	100
Projection dim	100
No. LSTMs	2

Table A.5: Hyperparameters of the topological field labeler in section 6.3.2

A.6 PP Attachment Disambiguation System: PP-BIAFFINE

Hyperparameter	**Value**
Lexical	
Word cutoff	2
Lowercase	True
Training	
Loss	Cross entropy
Optimizer	Adam
Learning rate	0.002
Decay rate	0.75
Decay step	5000
Batch size	5000 (words)
Max epoch	50

Hyperparameter	**Value**
No. train buckets	40
No. dev buckets	10
Regularization	
Max norm	5.0
Input dropout	0.33
Layer dropout	0.33
Recurrent dropout	0.33
L2	0
Neural Network	
Word emb dim	100
Tag emb dim	100
LSTM hidden dim	400
Projection dim	500
No. LSTMs	3

Table A.6: Hyperparameters of the PP attachment disambiguation system PP-BIAFFINE (section 6.4)

A.7 Neural Network Rerankers

A.7.1 Down-Sampling k-Best List

In order to maintain an efficient running time for our discriminative rerankers without scarifying the diversity we get from the k-best list, we apply down-sampling for each training instance. Namely, in each step, we use only 10 randomly selected trees (when $k > 10$) in addition to the gold tree to back-propagate.

We use the codes provided by Le and Zuidema (2014)[1] to train all IORNN rerankers with default hyperparameters for English, German and Czech data. The default number of training epochs is set to 50. Due to the time limit, we could only train the model for Czech which is the largest of our data sets up to 27 epochs, which took 15 days on a CPU. The program processes a single sentence at a time rather than batching or multithreading.

[1]https://github.com/lephong/iornn-depparse

A.7.2 RCNN

Hyperparameter	**Value**
Lexical	
Word cutoff	2
Lowercase	True
Training	
Loss	Average margin
Structured margin	False
Margin discount κ	1.0 (not used)
Optimizer	Adam
Learning rate	0.001
Decay rate	0.8
Decay step	8000
Batch size	16
Max epoch	20
Regularization	
Max norm	5.0
Input dropout	0.33
L2	0
Neural Network	
Word emb dim	50
Distance emb dim	25
RCNN hidden dim	50

Table A.7: Hyperparameters of the best RCNN re-ranking model for English PTB

A.7.3 RCNN-shared

Hyperparameter	**Value**
Lexical	
Word cutoff	2
Lowercase	True (En)
	False (De, Cs)
Training	

Hyperparameter	**Value**
Loss	Average margin
Structured margin	False
Margin discount η	1.0 (not used)
Optimizer	Adam
Learning rate	0.001
Decay rate	0.8
Decay step	16000
Batch size	16
Regularization	
Max norm	5.0
Input dropout	0.33
Dropout	0.33
Recurrent dropout	0.33
L2	0
Neural Network	
Word emb dim	50
POS emb dim	50
LSTM hidden dim	100
RCNN hidden dim	200
No. LSTMs	2

Table A.8: Hyperparameters of the best RCNN-shared re-ranking models

A.7.4 GCN

Hyperparameter	**Value**
Lexical	
Word cutoff	2
Lowercase	True (En)
	False (De, Cs)
Training	
Loss	Average margin
Structured margin	False
Margin discount η	1.0 (not used)

Hyperparameter	**Value**
Optimizer	Adam
Learning rate	0.001
Decay rate	0.8
Decay step	16000
Batch size	16
Max epoch	60
Regularization	
Max norm	5.0
Input dropout	0.33
Dropout	0.33
Recurrent dropout	0.33
L2	0
Neural Network	
Word emb dim	50
POS emb dim	50
LSTM hidden dim	100
GCN hidden dim	200
No. LSTMs	1
No. GCNs	3

Table A.9: Hyperparameters of the best GCN re-ranking models

A.7.5 Training

All discriminative rerankers are trained using Adam optimization (Kingma and Ba, 2015) with an initial learning rate of 0.001. The learning rate decays exponentially with a rate of 0.8.

A.7.6 Mixture Reranker

For the IORNN reranker, we use the tool provided in the repository instead of ours. The authors do logarithmic scaling for the score of the reranker in the mixture model combination:

$$s(x, y) = \alpha \log s_r(x, y, \Theta) + (1 - \alpha)s_b(x, y)$$

For all discriminative rerankers, in the experiment with the English data, we do logarithmic scaling for the score of the base parser in the mixture model combination:

$$s(x, y) = \alpha s_r(x, y, \Theta) + (1 - \alpha) \log s_b(x, y)$$

In the experiments with German and Czech data, we do not scale the score of the base parser and use equation 7.26.

APPENDIX B

Complete Results

B.1 Neural Network Rerankers

Table B.1: UAS of the GCN mixture model (table 7.8) on the PTB test set ($k = 64$) based on the POS tag of the modifier words

POS	Total	Baseline	GCNs	Δ
”	511	88.26	90.22	1.96
#	5	100.00	100.00	0.00
$	375	92.00	92.53	0.53
,	3,062	81.55	82.85	1.30
.	2,363	95.22	95.68	0.46
:	324	72.53	75.00	2.47
“	531	79.47	81.17	1.70
CC	1,367	87.56	88.81	1.25
CD	1,967	93.04	92.98	−0.06
DT	4,830	96.92	97.41	0.49
EX	60	95.00	95.00	0.00
FW	5	40.00	80.00	40.00
IN	5,974	85.99	87.11	1.12
JJ	3,573	93.45	94.12	0.67
JJR	203	81.77	82.76	0.99
JJS	125	94.40	96.00	1.60
-LRB-	72	88.89	87.50	−1.39
MD	584	93.15	94.52	1.37

Continued on next page

Table B.1 (continued): UAS of the GCN mixture model (table 7.8) on the PTB test set ($k = 64$) based on the POS tag of the modifier words

POS	Total	Baseline	GCNs	Δ
NN	7,634	93.04	94.26	1.22
NNP	5,737	95.47	96.23	0.76
NNPS	150	95.33	97.33	2.00
NNS	3,517	91.56	92.78	1.22
PDT	11	81.82	81.82	0.00
POS	555	95.32	96.04	0.72
PRP	1,049	97.62	98.67	1.05
PRP$	512	96.48	97.46	0.98
RB	1,939	83.75	86.59	2.84
RBR	97	79.38	74.23	−5.15
RBS	32	90.62	93.75	3.13
RP	150	96.00	96.67	0.67
-RRB-	72	84.72	84.72	0.00
SYM	1	100.00	100.00	0.00
TO	1,240	93.23	94.68	1.45
UH	8	50.00	50.00	0.00
VB	1,571	93.06	94.53	1.47
VBD	1,809	94.75	95.74	0.99
VBG	877	85.29	87.34	2.05
VBN	1,232	92.94	93.91	0.97
VBP	791	91.53	92.92	1.39
VBZ	1,229	93.73	94.06	0.33
WDT	276	86.96	87.32	0.36
WP	111	88.29	87.39	−0.90
WP$	21	90.48	85.71	−4.77
WRB	132	77.27	78.79	1.52

Table B.2: Labeled F1 of different dependency types from the German SPRML test set. *Total* is the number of instances in the gold file. RCNNs and GCNs are the results from the best mixture models of RCNN-shared ($k_{test} = 10$) and GCN ($k_{test} = 50$) from table 7.11.

Label	Total	Baseline	RCNNs	Δ_{RCNNs}	GCNs	Δ_{GCNs}
–	18,079	87.53	87.34	-0.19	87.43	-0.10
AC	127	91.13	91.56	0.43	89.60	-1.53
ADC	4	85.71	85.71	0.00	85.71	0.00
AG	2,241	93.43	93.07	-0.36	93.20	-0.24
AMS	75	87.90	88.32	0.42	89.47	1.57
APP	461	53.37	49.06	-4.31	50.52	-2.86
AVC	4	25.00	44.44	19.44	25.00	0.00
CC	238	72.46	74.00	1.54	74.53	2.07
CD	2,231	88.30	88.40	0.10	87.67	-0.63
CJ	3,446	81.78	82.46	0.68	82.50	0.72
CM	298	85.23	85.09	-0.14	84.60	-0.63
CP	788	94.88	94.70	-0.19	94.89	0.00
CVC	71	62.69	63.70	1.02	64.00	1.31
DA	568	67.25	67.57	0.32	68.40	1.15
DM	18	46.67	46.67	0.00	46.67	0.00
EP	186	84.77	85.32	0.55	85.32	0.55
JU	228	87.39	85.84	-1.54	86.74	-0.64
MNR	2,554	69.24	69.27	0.03	69.33	0.09
MO	12,102	78.84	78.56	-0.28	79.31	0.47
NG	514	78.02	78.02	0.00	77.57	-0.46
NK	27,906	97.51	97.51	0.00	97.46	-0.05
NMC	321	94.79	94.30	-0.48	94.17	-0.61
OA	3,184	82.84	82.98	0.14	84.76	1.91
OA2	5	NaN	NaN	NaN	NaN	NaN
OC	3,652	89.93	90.37	0.44	90.78	0.85
OG	21	48.49	48.49	0.00	52.94	4.45
OP	646	67.55	67.39	-0.17	67.11	-0.44
PAR	429	34.63	34.30	-0.33	35.25	0.62
PD	1,045	78.20	77.80	-0.40	79.63	1.42
PG	388	85.03	84.81	-0.22	84.53	-0.50

Continued on next page

Table B.2 (continued): Labeled F1 of different dependency types from the German SPRML test set. *Total* is the number of instances in the gold file. RCNNs and GCNs are the results from the best mixture models of RCNN-shared ($k_{test} = 10$) and GCN ($k_{test} = 50$) from table 7.11.

Label	Total	Baseline	RCNNs	Δ_{RCNNs}	GCNs	Δ_{GCNs}
PH	30	65.39	65.39	0.00	77.78	12.39
PM	432	98.96	99.08	0.11	99.42	0.46
PNC	1,246	85.49	85.80	0.31	85.67	0.18
RC	765	74.66	75.59	0.93	76.05	1.39
RE	268	74.46	76.46	2.00	76.63	2.16
RS	35	20.00	26.67	6.67	28.58	8.58
SB	6,638	89.20	89.25	0.06	90.67	1.48
SBP	156	75.78	76.40	0.62	78.71	2.94
SVP	499	91.92	92.13	0.20	91.65	−0.27
UC	90	24.35	25.01	0.65	23.00	−1.35
VO	15	NaN	NaN	NaN	NaN	NaN

Table B.3: Labeled F1 of different dependency types from the Czech UD test set. *Total* is the number of instances in the gold file. RCNNs and GCNs are the results from the best mixture models of RCNN-shared ($k_{test} = 50$) and GCN ($k_{test} = 50$) from table 7.12.

Label	Total	Baseline	RCNNs	Δ_{RCNNs}	GCNs	Δ_{GCNs}
acl	2,571	73.65	74.82	1.17	75.75	2.10
advcl	1,294	64.15	63.64	−0.51	65.94	1.79
advmod	7,272	87.59	87.21	−0.38	87.33	−0.26
advmod:emph	2,658	81.29	80.63	−0.66	80.80	−0.49
amod	17,810	97.99	97.92	−0.07	97.98	−0.01
appos	823	49.78	49.91	0.13	50.16	0.38
aux	1,803	97.15	97.34	0.19	97.15	0.00
aux:pass	674	88.78	88.86	0.08	88.79	0.01
case	16,479	98.47	98.39	−0.08	98.45	−0.03
cc	6,070	90.30	90.19	−0.12	90.42	0.11
ccomp	1,168	79.98	80.16	0.18	81.37	1.39
compound	212	82.84	80.58	−2.27	81.45	−1.40
conj	7,581	74.15	74.80	0.65	75.38	1.23
cop	2,576	88.55	88.48	−0.07	88.38	−0.17
csubj	617	81.82	81.82	−0.01	83.87	2.05
csubj:pass	50	33.71	31.46	−2.25	34.15	0.44
dep	1,563	52.40	52.80	0.39	52.25	−0.16
det	3,496	97.58	97.58	0.00	97.62	0.04
det:numgov	117	94.83	94.07	−0.76	93.22	−1.60
det:nummod	64	95.31	95.24	−0.07	96.83	1.52
discourse	45	10.17	10.53	0.36	7.40	−2.77
expl:pass	559	75.05	73.99	−1.06	73.52	−1.53
expl:pv	1,946	88.30	88.05	−0.24	88.11	−0.18
fixed	566	87.60	87.45	−0.15	87.54	−0.05
flat	2,295	93.87	93.91	0.04	94.03	0.17
flat:foreign	327	75.16	76.66	1.50	75.00	−0.16
iobj	588	60.92	60.43	−0.49	63.14	2.22
mark	3,429	94.25	94.12	−0.13	94.04	−0.20
nmod	17,145	87.66	87.43	−0.24	87.58	−0.08
nsubj	10,107	91.50	91.47	−0.03	91.96	0.46

Continued on next page

Table B.3 (continued): Labeled F1 of different dependency types from the Czech UD test set. *Total* is the number of instances in the gold file. RCNNs and GCNs are the results from the best mixture models of RCNN-shared ($k_{test} = 50$) and GCN ($k_{test} = 50$) from table 7.12.

Label	Total	Baseline	RCNNs	Δ_{RCNNs}	GCNs	Δ_{GCNs}
nsubj:pass	886	70.19	70.27	0.08	72.02	1.82
nummod	2,168	90.32	90.25	−0.07	90.28	−0.04
nummod:gov	830	92.64	92.41	−0.23	92.57	−0.07
obj	7,592	90.10	90.09	−0.01	90.64	0.54
obl	10,009	80.04	79.70	−0.35	79.97	−0.07
obl:agent	67	60.00	59.58	−0.42	57.35	−2.65
obl:arg	2,058	76.73	76.61	−0.11	76.29	−0.44
orphan	418	23.31	21.98	−1.33	29.05	5.73
parataxis	233	45.98	43.18	−2.80	46.76	0.78
punct	25,416	89.36	89.39	0.03	89.38	0.01
root	10,148	95.25	95.26	0.01	95.26	0.01
vocative	12	12.50	14.28	1.78	15.38	2.88
xcomp	2,176	87.45	87.65	0.20	88.16	0.70

APPENDIX C Resources

C.1 Chapter 4

- The German dataset from the SPMRL 2014 shared task data (Seddah et al., 2014) is available at https://www.spmrl.org/spmrl2014-sharedtask.html.
- The IMS Splitter (Weller and Heid, 2012) is downloaded from http://www.ims.uni-stuttgart.de/institut/mitarbeiter/wellermn/tools.html. Input information for the splitter is extracted from the SdeWac corpus (Faaß and Eckart, 2013) and available at https://wacky.sslmit.unibo.it/.
- The language models are trained by the code from https://github.com/claravania/subword-lstm-lm on the preprocessed German Wikipedia text dump at https://sites.google.com/site/rmyeid/projects/polyglot.
- The data repository at https://doi.org/10.11588/data/BPWWJL contains the code of the dependency parsers, processed data and pre-trained parsing models used in the chapter.

C.2 Chapter 5

- The English Penn Treebank (Marcus et al., 1993) is available at https://catalog.ldc.upenn.edu/LDC99T42. The POS tags are assigned using the Stanford POS tagger (Toutanova et al., 2003) at https://nlp.stanford.edu/software/tagger.shtml. Constituency trees are converted to Stanford basic dependencies (De Marneffe et al., 2006) using the Stanford parser version

3.3.0 at https://nlp.stanford.edu/software/lex-parser.shtml.

- The German and Czech data from the CoNLL-X Shared Task (Buchholz and Marsi, 2006) are available at https://catalog.ldc.upenn.edu/LDC2015T11 and https://catalog.ldc.upenn.edu/LDC2015T12.
- The German data from the SPMRL 2014 shared task data (Seddah et al., 2014) are available at https://www.spmrl.org/spmrl2014-sharedtask.html.
- The data repository at https://doi.org/10.11588/data/BPWWJL contains the code of the LSTM labelers, processed data and pre-trained parsing and labeling models used in the chapter.

C.3 Chapter 6

- The PP attachment dataset (de Kok et al., 2017a) for the reproduction work in section 6.3.1 is provided by the authors. The word and tag embeddings are downloaded from http://hdl.handle.net/11022/0000-0000-8507-2.
- The articles from the *taz* newspapers that are used to compute the auxiliary distributions are from the TüPP-D/Z corpus, which can be requested from https://uni-tuebingen.de/fakultaeten/philosophische-fakultaet/fachbereiche/neuphilologie/seminar-fuer-sprachwissenschaft/arbeitsbereiche/allg-sprachwissenschaft-computerlinguistik/ressourcen/corpora/tuepp-dz/.
- We use version 11.0 of the Tüba-D/Z corpus to train the topological field labeler, which is available from https://uni-tuebingen.de/fakultaeten/philosophische-fakultaet/fachbereiche/neuphilologie/seminar-fuer-sprachwissenschaft/arbeitsbereiche/allg-sprachwissenschaft-computerlinguistik/ressourcen/corpora/tueba-dz/.
- The BERT$_{\text{BASE}}$ Multilingual Cased language model is download from https://github.com/google-research/bert.
- The German PP attachment disambiguation dataset is published at the data repository https://doi.org/10.11588/data/NB46XR.

- The code, pre-trained models and resources to reproduce the experiments in this chapter are available at different data repositories. The code of each tool is also publicly available at a separated repository. More specifically:
  - Tool for extracting PP attachment disambiguation datasets:
    https://doi.org/10.11588/data/RHD3KS
    https://github.com/bichngocdo/pp-attachment-candidate-extraction
  - PP attachment disambiguation systems:
    https://doi.org/10.11588/data/DKWKGJ
    https://github.com/bichngocdo/pp-disambiguation-ranking
    https://github.com/bichngocdo/biaffine-pp-disambiguation
  - Neural dependency parser with biaffine attention and BERT embeddings:
    https://doi.org/10.11588/data/0U6IWL
    https://github.com/bichngocdo/bert-biaffine-parser
  - Topological labeler for German:
    https://doi.org/10.11588/data/YYNQFF
    https://github.com/bichngocdo/topological-field-labeler

C.4 Chapter 7

- The English Penn Treebank data are downloaded from https://github.com/lianghuang3/lineardpparser/tree/master/data. Extracting the top *k* parses from the forests is done by using a fork from the original repository: https://github.com/bichngocdo/lineardpparser.

- The GloVe word embeddings for English are downloaded from https://nlp.stanford.edu/data/glove.6B.zip.

- The German data from the SPMRL 2014 shared task data (Seddah et al., 2014) are available at https://www.spmrl.org/spmrl2014-sharedtask.html.

- The Czech UD Treebank is downloaded from https://github.com/UniversalDependencies/UD_Czech-PDT/tree/fd9517f8305ac997fd57e20760d9ecdb9d76bf2b.

- The fastText Czech word embeddings are from https://fasttext.cc/docs/en/crawl-vectors.html (*text* version). Dimension reduction is done with the tool from https://github.com/vyraun/Half-Size.
- Extracting top k parses for German and Czech is done using a version of the graph-based parser in the MATE tools provided by the authors.
- The dependency reranking datasets for 3 languages: English, German, and Czech are published at the data repository https://doi.org/10.11588/data/E5NOYH.
- The neural reranking code, pre-trained models and resources to reproduce the experiments in this chapter are available at the data repository https://doi.org/10.11588/data/NNGPQZ.
- The code for different neural rerankers is also publicly available at https://github.com/bichngocdo/neural-tree-reranking.

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

ANN	Artificial neural network
BPTT	Backpropagation through time
CBOW	Continuous bag-of-words
CKY	Cocke-Kasami-Younger
CNN	Convolutional neural network
CRF	Conditional random field
GCN	Graph convolutional network
GRU	Gated recurrent unit
IORNN	Inside-outside recursive neural network
LAS	Labeled attachment score
LSTM	Long short-term memory
MEE	Maximum entropy estimation
MF	*Mittefeld*, middle field
MIRA	Margin infused relaxed algorithm
MLE	Maximum likelihood estimation
MLP	Multilayer perceptron
MRL	Morphologically rich language
MST	Maximum spanning tree

NF	*Nachfeld*, final field
NLP	Natural language processing
NLU	Natural language processing
OOV	Out-of-vocabulary
POS	Part-of-speech
PP	Prepositional phrase
PTB	Penn Treebank
RCNN	Recurrent convolutional neural network
RNN	Recurrent neural network
SVM	Support vector machine
UAS	Unlabeled attachment score
UD	Universal Dependencies
VF	*Vorfeld*, initial field
WSJ	Wall Street Journal

Bibliography

Eneko Agirre, Timothy Baldwin and David Martinez (2008). "Improving Parsing and PP Attachment Performance with Sense Information". In: *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Columbus, Ohio: Association for Computational Linguistics, pp. 317–325 (see pages 96, 98, 99, 114).

Rami Al-Rfou, Bryan Perozzi and Steven Skiena (2013). "Polyglot: Distributed Word Representations for Multilingual NLP". In: *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 183–192 (see page 79).

Martha A. Alegre, Josep M. Sopena and Agusti Lloberas (1999). "PP-Attachment: A Committee Machine Approach". In: *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora* (see pages 97, 98).

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov and Michael Collins (2016). "Globally Normalized Transition-based Neural Networks". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 2442–2452 (see pages 64, 67, 68).

Michaela Atterer and Hinrich Schütze (2007). "Prepositional Phrase Attachment without Oracles". In: *Computational Linguistics* 33.4, pp. 469–476. DOI: 10.1162/coli.2007.33.4.469 (see pages 93, 98, 113, 115).

Dzmitry Bahdanau, Kyunghyun Cho and Yoshua Bengio (2015). "Neural Machine Translation by Jointly Learning to Align and Translate". In: *Proceedings of the 3rd International Conference on Learning Representations*. San Diego, CA, USA. arXiv: 1409.0473 [cs.CL] (see pages 19, 20, 30, 66, 96).

Collin F. Baker, Charles J. Fillmore and John B. Lowe (1998). "The Berkeley FrameNet Project". In: *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics,* *Volume 1*. Montreal, Quebec, Canada: Association for Computational Linguistics, pp. 86–90. DOI: 10.3115/980845.980860 (see page 97).

Miguel Ballesteros, Chris Dyer and Noah A. Smith (2015). “Improved Transition-based Parsing by Modeling Characters instead of Words with LSTMs”. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 349–359. DOI: 10.18653/v1/D15-1041 (see pages 28, 69, 71).

Miguel Ballesteros, Yoav Goldberg, Chris Dyer and Noah A. Smith (2016). “Training with Exploration Improves a Greedy Stack LSTM Parser”. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, pp. 2005–2010. DOI: 10.18653/v1/D16-1211 (see page 52).

Marco Baroni, Silvia Bernardini, Adriano Ferraresi and Eros Zanchetta (2009). “The WaCky Wide Web: A Collection of Very Large Linguistically Processed Web-Crawled Corpora”. In: *Language Resources and Evaluation* 43.3, pp. 209–226. ISSN: 1574-0218. DOI: 10.1007/s10579-009-9081-4 (see page 74).

Eduard Bejček, Eva Hajičová, Jan Hajič, Pavlína Jínová, Václava Kettnerová, Veronika Kolářová, Marie Mikulová, Jiří Mírovský, Anna Nedoluzhko, Jarmila Panevová, Lucie Poláková, Magda Ševčíková, Jan Štěpánek and Šárka Zikánová (2013). *Prague Dependency Treebank 3.0*. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University (see page 136).

Yonatan Belinkov, Tao Lei, Regina Barzilay and Amir Globerson (2014). “Exploring Compositional Architectures and Word Vector Representations for Prepositional Phrase Attachment”. In: *Transactions of the Association for Computational Linguistics* 2, pp. 561–572. DOI: 10.1162/tacl_a_00203 (see pages 93, 94, 96–99).

Yoshua Bengio, Réjean Ducharme and Pascal Vincent (2001). “A Neural Probabilistic Language Model”. In: *Advances in Neural Information Processing Systems 13 (NIPS 2000)*. Ed. by T. Leen, T. Dietterich and V. Tresp. Vol. 13. MIT Press, pp. 932–938 (see pages 24, 25).

Anders Björkelund, Özlem Çetinoğlu, Agnieszka Faleńska, Richárd Farkas, Thomas Mueller, Wolfgang Seeker and Zsolt Szántó (2014). “Introducing the IMS-Wrocław-Szeged-CIS entry at the SPMRL 2014 Shared Task: Reranking and Morpho-syntax meet Unlabeled Data”. In: *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*. Dublin, Ireland: Dublin City University, pp. 97–102 (see pages 86, 87).

Anders Björkelund and Joakim Nivre (2015). “Non-Deterministic Oracles for Unrestricted Non-Projective Transition-Based Dependency Parsing”. In: *Proceedings of* *the 14th International Conference on Parsing Technologies*. Bilbao, Spain: Association for Computational Linguistics, pp. 76–86. DOI: 10.18653/v1/W15-2210 (see page 50).

Don Blaheta and Eugene Charniak (2000). "Assigning Function Tags to Parsed Text". In: *1st Meeting of the North American Chapter of the Association for Computational Linguistics* (see page 82).

Bernd Bohnet (2010). "Top Accuracy and Fast Dependency Parsing is not a Contradiction". In: *Proceedings of the 23rd International Conference on Computational Linguistics*. Beijing, China: Coling 2010 Organizing Committee, pp. 89–97 (see pages 59, 106, 136).

Bernd Bohnet and Jonas Kuhn (2012). "The Best of Both Worlds – A Graph-based Completion Model for Transition-based Parsers". In: *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*. Avignon, France: Association for Computational Linguistics, pp. 77–87 (see page 61).

Bernd Bohnet and Joakim Nivre (2012). "A Transition-Based System for Joint Part-of-Speech Tagging and Labeled Non-Projective Dependency Parsing". In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Jeju Island, Korea: Association for Computational Linguistics, pp. 1455–1465 (see page 41).

Piotr Bojanowski, Edouard Grave, Armand Joulin and Tomas Mikolov (2017). "Enriching Word Vectors with Subword Information". In: *Transactions of the Association for Computational Linguistics* 5, pp. 135–146. DOI: 10.1162/tacl_a_00051 (see pages 28, 138).

Jan Botha and Phil Blunsom (2014). "Compositional Morphology for Word Representations and Language Modelling". In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. Bejing, China: PMLR, pp. 1899–1907 (see page 71).

Gerlof Bouma (2009). "Normalized (Pointwise) Mutual Information in Collocation Extraction". In: *Proceedings of the International Conference of the German Society for Computational Linguistics and Language Technology (GSCL 2009)*. Postdam, Germany, pp. 31–40 (see page 103).

Eric Brill and Philip Resnik (1994). "A Rule-Based Approach to Prepositional Phrase Attachment Disambiguation". In: *Proceedings of the 15th International Conference on Computational Linguistics, Volume 2* (see pages 91, 93, 96, 98).

Sabine Buchholz and Erwin Marsi (2006). "CoNLL-X Shared Task on Multilingual Dependency Parsing". In: *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*. New York City, USA: Association for Computational Linguistics, pp. 149–164 (see pages 85, 176).

Paolo M. Camerini, Luigi Fratta and Francesco Maffioli (1980). "The K Best Spanning Arborescences of a Network". In: *Networks* 10.2, pp. 91–109 (see page 59).

Kris Cao and Marek Rei (2016). "A Joint Model for Word Embedding and Word Morphology". In: *Proceedings of the 1st Workshop on Representation Learning for NLP*. Berlin, Germany: Association for Computational Linguistics, pp. 18–26. DOI: 10.18653/v1/W16-1603 (see pages 71, 78).

Xavier Carreras (2007). "Experiments with a Higher-Order Projective Dependency Parser". In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. Prague, Czech Republic: Association for Computational Linguistics, pp. 957–961 (see pages 57–59).

Eugene Charniak and Mark Johnson (2005). "Coarse-to-Fine n-Best Parsing and MaxEnt Discriminative Reranking". In: *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*. Ann Arbor, Michigan: Association for Computational Linguistics, pp. 173–180. DOI: 10.3115/1219840.1219862 (see page 126).

Danqi Chen and Christopher D. Manning (2014). "A Fast and Accurate Dependency Parser using Neural Networks". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. Doha, Qatar: Association for Computational Linguistics, pp. 740–750 (see pages 22, 25, 62–64, 66, 68, 72, 97, 179).

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk and Yoshua Bengio (2014). "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. Doha, Qatar: Association for Computational Linguistics, pp. 1724–1734. DOI: 10.3115/v1/D14-1179 (see pages 17, 66).

Noam Chomsky (1957). *Syntactic structures* (see page 35).

Grzegorz Chrupała and Josef van Genabith (2006). "Using Machine-Learning to Assign Function Labels to Parser Output for Spanish". In: *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*. Sydney, Australia: Association for Computational Linguistics, pp. 136–143 (see page 82).

Yoeng-Jin Chu and Tseng-Hong Liu (1965). "On the Shortest Arborescence of a Directed Graph". In: *Scientia Sinica* 14, pp. 1396–1400 (see page 53).

Jasmine Collins, Jascha Sohl-Dickstein and David Sussillo (2017). "Capacity and Trainability in Recurrent Neural Networks". In: *Proceedings of the 5th International Conference on Learning Representations*. Toulon, France (see page 18).

Michael Collins (2002). "Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms". In: *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pp. 1–8. DOI: `10.3115/1118693.1118694` (see pages 47, 56).

Michael Collins and James Brooks (1995). "Prepositional Phrase Attachment through a Backed-off Model". In: *Proceedings of the Third Workshop on Very Large Corpora* (see pages 95, 98).

Michael Collins and Terry Koo (2005). "Discriminative Reranking for Natural Language Parsing". In: *Computational Linguistics* 31.1, pp. 25–70. DOI: `10.1162/0891201053630273` (see pages 48, 126).

Michael Collins and Brian Roark (2004). "Incremental Parsing with the Perceptron Algorithm". In: *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*. Barcelona, Spain, pp. 111–118. DOI: `10.3115/1218955.1218970` (see page 48).

Ronan Collobert and Jason Weston (2008). "A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning". In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: Association for Computing Machinery, 160–167. ISBN: 9781605582054. DOI: `10.1145/1390156.1390177` (see pages 22, 25, 26).

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu and Pavel Kuksa (2011). "Natural Language Processing (Almost) from Scratch". In: *Journal of Machine Learning Research* 12.76, pp. 2493–2537 (see pages 22, 25).

Çağrı Çöltekin, Ben Campbell, Erhard Hinrichs and Heike Telljohann (2017). "Converting the TüBa-D/Z Treebank of German to Universal Dependencies". In: *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*. Gothenburg, Sweden: Association for Computational Linguistics, pp. 27–37 (see page 109).

Ryan Cotterell and Hinrich Schütze (2015). "Morphological Word-Embeddings". In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Denver, Colorado: Association for Computational Linguistics, pp. 1287–1292. DOI: `10.3115/v1/N15-1140` (see page 71).

Koby Crammer and Yoram Singer (2003). "Ultraconservative Online Algorithms for Multiclass Problems". In: *Journal of Machine Learning Research* 3.Jan, pp. 951–991 (see page 56).

James Cross and Liang Huang (2016). "Incremental Parsing with Minimal Features Using Bi-Directional LSTM". In: *Proceedings of the 54th Annual Meeting of the*

Association for Computational Linguistics (Volume 2: Short Papers). Berlin, Germany: Association for Computational Linguistics, pp. 32–37. DOI: 10.18653/v1/P16-2006 (see pages 66–68).

George Cybenko (1989). "Approximation by Superpositions of a Sigmoidal Function". In: *Mathematics of Control, Signals and Systems* 2.4, pp. 303–314 (see page 14).

Pradeep Dasigi, Waleed Ammar, Chris Dyer and Eduard Hovy (2017). "Ontology-Aware Token Embeddings for Prepositional Phrase Attachment". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 2089–2098. DOI: 10.18653/v1/P17-1191 (see pages 96–99).

Daniël de Kok (2014). "TüBa-D/W: A Large Dependency Treebank for German". In: *Proceedings of the 13th International Workshop on Treebanks and Linguistic Theories (TLT13)*. Tübingen, Germany, p. 271 (see page 103).

Daniël de Kok, Corina Dima, Jianqiang Ma and Erhard Hinrichs (2017a). "Extracting a PP Attachment Data Set from a German Dependency Treebank Using Topological Fields". In: *Proceedings of the 15th International Workshop on Treebanks and Linguistic Theories (TLT15)*. Bloomington, IN, USA (see pages 93, 99, 101, 102, 104, 105, 107, 110, 111, 114, 176).

Daniël de Kok and Erhard Hinrichs (2016). "Transition-Based Dependency Parsing with Topological Fields". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 1–7. DOI: 10.18653/v1/P16-2001 (see page 108).

Daniël de Kok, Jianqiang Ma, Corina Dima and Erhard Hinrichs (2017b). "PP Attachment: Where do We Stand?" In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Valencia, Spain: Association for Computational Linguistics, pp. 311–317 (see pages xi, 94, 97–99, 102, 104–109, 111, 114, 182).

Marie-Catherine De Marneffe, Bill MacCartney and Christopher D. Manning (2006). "Generating Typed Dependency Parses from Phrase Structure Parses". In: *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*. Vol. 6. 2006. Genoa, Italy: European Language Resources Association (ELRA), pp. 449–454 (see pages 85, 175).

Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and*

Short Papers). Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4171–4186. DOI: 10.18653/v1/N19-1423 (see pages 31–33, 69, 120).

Corina Dima (2015). "Reverse-engineering Language: A Study on the Semantic Compositionality of German Compounds". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 1637–1642. DOI: 10.18653/v1/D15-1188 (see page 71).

Corina Dima and Erhard Hinrichs (2015). "Automatic Noun Compound Interpretation using Deep Neural Networks and Word Embeddings". In: *Proceedings of the 11th International Conference on Computational Semantics*. London, UK: Association for Computational Linguistics, pp. 173–183 (see page 71).

Bich-Ngoc Do and Ines Rehbein (2017). "Evaluating LSTM Models for Grammatical Function Labelling". In: *Proceedings of the 15th International Conference on Parsing Technologies*. Pisa, Italy: Association for Computational Linguistics, pp. 128–133 (see page 9).

Bich-Ngoc Do and Ines Rehbein (2020a). "Neural Reranking for Dependency Parsing: An Evaluation". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, pp. 4123–4133. DOI: 10.18653/v1/2020.acl-main.379 (see page 9).

Bich-Ngoc Do and Ines Rehbein (2020b). "Parsers Know Best: German PP Attachment Revisited". In: *Proceedings of the 28th International Conference on Computational Linguistics*. Barcelona, Spain (Online): International Committee on Computational Linguistics, pp. 2049–2061. DOI: 10.18653/v1/2020.coling-main.185 (see page 9).

Bich-Ngoc Do, Ines Rehbein and Anette Frank (2017). "What Do We Need to Know about an Unknown Word When Parsing German". In: *Proceedings of the First Workshop on Subword and Character Level Models in NLP*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 117–123. DOI: 10.18653/v1/W17-4117 (see page 9).

Cicero dos Santos and Bianca Zadrozny (2014). "Learning Character-level Representations for Part-of-Speech Tagging". In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. Bejing, China: JMLR Workshop and Conference Proceedings, pp. 1818–1826 (see pages 28, 71).

Timothy Dozat and Christopher D. Manning (2017). "Deep Biaffine Attention for Neural Dependency Parsing". In: *Proceedings of the 5th International Conference on Learning Representations*. Toulon, France (see pages 67–69, 91, 105, 107, 108, 118, 125, 160).

Timothy Dozat, Peng Qi and Christopher D. Manning (2017). “Stanford’s Graph-based Neural Dependency Parser at the CoNLL 2017 Shared Task”. In: *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Vancouver, Canada: Association for Computational Linguistics, pp. 20–30. DOI: 10.18653/v1/K17-3002 (see page 123).

Erich Drach (1937). *Grundgedanken der deutschen Satzlehre*. Frankfurt: Diesterweg (see page 100).

John Duchi, Elad Hazan and Yoram Singer (2011). “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”. In: *Journal of Machine Learning Research* 12.61, pp. 2121–2159 (see page 103).

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews and Noah A. Smith (2015). “Transition-Based Dependency Parsing with Stack Long Short-Term Memory”. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 334–343 (see pages 65, 67, 68, 180).

Jack Edmonds (1967). “Optimum Branchings”. In: *Journal of Research of the National Bureau of Standards* 71B.4, pp. 233–240 (see page 53).

Jason M. Eisner (1996). “Three New Probabilistic Models for Dependency Parsing: An Exploration”. In: *Proceedings of the 16th International Conference on Computational Linguistics, Volume 1* (see pages 55, 128).

Jeffrey L. Elman (1990). “Finding Structure in Time”. In: *Cognitive Science* 14.2, pp. 179–211. DOI: 10.1207/s15516709cog1402_1 (see page 15).

Gertrud Faaß and Kerstin Eckart (2013). “SdeWaC - A Corpus of Parsable Sentences from the Web”. In: *Language Processing and Knowledge in the Web: Proceedings of the 25th International Conference of the German Society for Computational Linguistics (GSCL 2013)*. Darmstadt, Germany: Springer, pp. 61–68 (see pages 74, 108, 137, 175).

Christiane Fellbaum, ed. (1998). *WordNet: An Electronic Lexical Database*. MIT Press. ISBN: 9780262061971 (see page 95).

Daniel Fernández-González and Carlos Gómez-Rodríguez (2019). “Left-to-Right Dependency Parsing with Pointer Networks”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 710–716. DOI: 10.18653/v1/N19-1076 (see page 69).

Kilian A. Foth and Wolfgang Menzel (2006). “The Benefit of Stochastic PP Attachment to a Rule-Based Parser”. In: *Proceedings of the 21st International Conference on*

Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics: Main Conference Poster Sessions. Sydney, Australia: Association for Computational Linguistics, pp. 223–230 (see page 98).

Dan Gillick, Cliff Brunk, Oriol Vinyals and Amarnag Subramanya (2016). "Multilingual Language Processing From Bytes". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, pp. 1296–1306. DOI: 10.18653/v1/N16-1155 (see page 71).

Goran Glavaš and Ivan Vulić (2020). *Is Supervised Syntactic Parsing Beneficial for Language Understanding? An Empirical Investigation*. arXiv: 2008.06788 [cs.CL] (see page 69).

Xavier Glorot, Antoine Bordes and Yoshua Bengio (2011). "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: JMLR Workshop and Conference Proceedings, pp. 315–323 (see page 13).

Yoav Goldberg (2017). "Neural Network Methods for Natural Language Processing". In: *Synthesis Lectures on Human Language Technologies* 10.1, pp. 1–309. DOI: 10.2200/S00762ED1V01Y201703HLT037 (see pages 11, 14, 23).

Yoav Goldberg and Joakim Nivre (2012). "A Dynamic Oracle for Arc-Eager Dependency Parsing". In: *Proceedings of COLING 2012*. Mumbai, India: The COLING 2012 Organizing Committee, pp. 959–976 (see page 52).

Yoav Goldberg and Joakim Nivre (2013). "Training Deterministic Parsers with Non-Deterministic Oracles". In: *Transactions of the Association for Computational Linguistics* 1, pp. 403–414. DOI: 10.1162/tacl_a_00237 (see pages 50, 51).

Yoav Goldberg, Francesco Sartorio and Giorgio Satta (2014). "A Tabular Method for Dynamic Oracles in Transition-Based Parsing". In: *Transactions of the Association for Computational Linguistics* 2, pp. 119–130. DOI: 10.1162/tacl_a_00170 (see page 52).

Ian Goodfellow, Yoshua Bengio and Aaron Courville (2016). *Deep Learning*. MIT Press (see page 11).

Kristina Gulordava and Paola Merlo (2016). "Multi-lingual Dependency Parsing Evaluation: a Large-scale Analysis of Word Order Properties using Artificial Data". In: *Transactions of the Association for Computational Linguistics* 4, pp. 343–356. DOI: 10.1162/tacl_a_00103 (see page 88).

Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel Bowman and Noah A. Smith (2018). "Annotation Artifacts in Natural Language Inference Data". In: *Proceedings of the 2018 Conference of the North American Chapter* *of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 107–112. DOI: 10.18653/v1/N18-2017 (see page 2).

Richard H. R. Hahnloser, Rahul Sarpeshkar, Misha A. Mahowald, Rodney J. Douglas and H. Sebastian Seung (2000). "Digital Selection and Analogue Amplification Coexist in a Cortex-Inspired Silicon Circuit". In: *Nature* 405.6789, pp. 947–951 (see page 103).

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue and Yi Zhang (2009). "The CoNLL-2009 Shared Task: Syntactic and Semantic Dependencies in Multiple Languages". In: *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL 2009): Shared Task*. Boulder, Colorado: Association for Computational Linguistics, pp. 1–18 (see page 106).

Keith Hall (2007). "K-Best Spanning Tree Parsing". In: *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*. Prague, Czech Republic: Association for Computational Linguistics, pp. 392–399 (see pages 59, 149).

Zellig S. Harris (1954). "Distributional Structure". In: *Word* 10.2–3, pp. 146–162. DOI: 10.1080/00437956.1954.11659520 (see page 24).

Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka and Richard Socher (2017). "A Joint Many-Task Model: Growing a Neural Network for Multiple NLP Tasks". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 1923–1933. DOI: 10.18653/v1/D17-1206 (see page 69).

Katsuhiko Hayashi, Taro Watanabe, Masayuki Asahara and Yuji Matsumoto (2011). "Third-order Variational Reranking on Packed-Shared Dependency Forests". In: *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*. Edinburgh, Scotland, UK.: Association for Computational Linguistics, pp. 1479–1488 (see page 127).

Benjamin Heinzerling and Michael Strube (2018). "BPEmb: Tokenization-free Pre-trained Subword Embeddings in 275 Languages". In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Miyazaki, Japan: European Language Resources Association (ELRA) (see page 28).

Donald Hindle and Mats Rooth (1993). "Structural Ambiguity and Lexical Relations". In: *Computational Linguistics* 19.1, pp. 103–120 (see pages 91, 93, 94, 98).

Erhard Hinrichs, Sandra Kübler, Karin Naumann, Heike Telljohann and Julia Trushkina (2004). "Recent Developments in Linguistic Annotations of the TüBa-D/Z

Treebank". In: *Proceedings of the Third Workshop on Treebanks and Linguistic Theories*. Tübingen, Germany, pp. 51–62 (see page 99).

Sepp Hochreiter and Jürgen Schmidhuber (1997). "Long Short-Term Memory". In: *Neural Comput.* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735 (see pages 17, 64, 73, 82, 97).

Tilman Höhle (1986). "Der Begriff 'Mittelfeld'. Anmerkungen über die Theorie der topologischen Felder". In: *Kontroversen alte und neue: Akten des 7. Internationalen Germanistenkongresses, Göttingen, 1985*. Ed. by Albrecht Schöne. Tübingen: Niemeyer, pp. 329–340 (see pages 100, 101).

Kurt Hornik (1991). "Approximation Capabilities of Multilayer Feedforward Networks". In: *Neural Networks* 4.2, pp. 251–257 (see page 14).

Liang Huang (2008). "Forest Reranking: Discriminative Parsing with Non-Local Features". In: *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Columbus, Ohio: Association for Computational Linguistics, pp. 586–594 (see page 127).

Liang Huang and David Chiang (2005). "Better k-Best Parsing". In: *Proceedings of the Ninth International Workshop on Parsing Technology*. Vancouver, British Columbia: Association for Computational Linguistics, pp. 53–64 (see page 127).

Liang Huang, Suphan Fayong and Yang Guo (2012). "Structured Perceptron with Inexact Search". In: *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Montréal, Canada: Association for Computational Linguistics, pp. 142–151 (see page 48).

Liang Huang and Kenji Sagae (2010). "Dynamic Programming for Linear-Time Incremental Parsing". In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Uppsala, Sweden: Association for Computational Linguistics, pp. 1077–1086 (see pages 135–137, 148).

Sergey Ioffe and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: JMLR Workshop and Conference Proceedings, pp. 448–456 (see pages 17, 103).

Ozan İrsoy and Claire Cardie (2014). "Opinion Mining with Deep Recurrent Neural Networks". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 720–728. DOI: 10.3115/v1/D14-1080 (see page 66).

Tao Ji, Yuanbin Wu and Man Lan (2019). "Graph-based Dependency Parsing with Graph Neural Networks". In: *Proceedings of the 57th Annual Meeting of the Associ-*

ation for Computational Linguistics. Florence, Italy: Association for Computational Linguistics, pp. 2475–2485. DOI: 10.18653/v1/P19-1237 (see page 69).

Valentin Jijkoun and Maarten de Rijke (2004). "Enriching the Output of a Parser Using Memory-based Learning". In: *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*. Barcelona, Spain, pp. 311–318. DOI: 10.3115/1218955.1218995 (see page 82).

Yoon Kim, Yacine Jernite, David Sontag and Alexander M. Rush (2016). "Character-Aware Neural Language Models". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI'16. Phoenix, Arizona, USA: AAAI Press, 2741–2749 (see page 78).

Diederik Kingma and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization". In: *Proceedings of the 3rd International Conference on Learning Representations*. San Diego, CA, USA (see pages 76, 86, 167).

Eliyahu Kiperwasser and Yoav Goldberg (2016a). "Easy-First Dependency Parsing with Hierarchical Tree LSTMs". In: *Transactions of the Association for Computational Linguistics* 4, pp. 445–461. DOI: 10.1162/tacl_a_00110 (see pages 18, 69).

Eliyahu Kiperwasser and Yoav Goldberg (2016b). "Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations". In: *Transactions of the Association for Computational Linguistics* 4, pp. 313–327. ISSN: 2307-387X (see pages 66–68).

Dan Klein and Christopher Manning (2004). "Corpus-Based Induction of Syntactic Structure: Models of Dependency and Constituency". In: *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*. Barcelona, Spain, pp. 478–485. DOI: 10.3115/1218955.1219016 (see page 57).

Manfred Klenner (2007). "Shallow Dependency Labeling". In: *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions*. Prague, Czech Republic: Association for Computational Linguistics, pp. 201–204 (see page 82).

Arne Köhn (2016). "Evaluating Embeddings using Syntax-based Classification Tasks as a Proxy for Parser Performance". In: *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*. Berlin, Germany: Association for Computational Linguistics, pp. 67–71. DOI: 10.18653/v1/W16-2512 (see page 78).

Dan Kondratyuk and Milan Straka (2019). "75 Languages, 1 Model: Parsing Universal Dependencies Universally". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, pp. 2779–2795. DOI: 10.18653/v1/D19-1279 (see page 69).

Terry Koo, Amir Globerson, Xavier Carreras and Michael Collins (2007). "Structured Prediction Models via the Matrix-Tree Theorem". In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. Prague, Czech Republic: Association for Computational Linguistics, pp. 141–150 (see page 57).

Marco Kuhlmann, Carlos Gómez-Rodríguez and Giorgio Satta (2011). "Dynamic Programming Algorithms for Transition-Based Dependency Parsers". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, OR, USA: Association for Computational Linguistics, pp. 673–682 (see pages 44, 52).

Jonathan K. Kummerfeld, David Hall, James R. Curran and Dan Klein (2012). "Parser Showdown at the Wall Street Corral: An Empirical Investigation of Error Types in Parser Output". In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Jeju Island, Korea: Association for Computational Linguistics, pp. 1048–1059 (see page 91).

Sandra Kübler, Ryan McDonald and Joakim Nivre (2009). "Dependency Parsing". In: *Synthesis Lectures on Human Language Technologies* 2.1, pp. 1–127. DOI: 10.2200/S00169ED1V01Y200901HLT002 (see pages 35, 54).

Phong Le and Willem Zuidema (2014). "The Inside-Outside Recursive Neural Network model for Dependency Parsing". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. Doha, Qatar: Association for Computational Linguistics, pp. 729–739. DOI: 10.3115/v1/D14-1081 (see pages 18, 125–130, 139, 164).

Omer Levy and Yoav Goldberg (2014). "Dependency-Based Word Embeddings". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Baltimore, Maryland: Association for Computational Linguistics, pp. 302–308 (see pages 27, 108, 137).

Jiwei Li, Thang Luong, Dan Jurafsky and Eduard Hovy (2015). "When Are Tree Structures Necessary for Deep Learning of Representations?" In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 2304–2314. DOI: 10.18653/v1/D15-1278 (see page 18).

Wang Ling, Chris Dyer, Alan W. Black and Isabel Trancoso (2015a). "Two/Too Simple Adaptations of Word2Vec for Syntax Problems". In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Denver, Colorado: Association for Computational Linguistics, pp. 1299–1304. DOI: 10.3115/v1/N15-1142 (see page 27).

Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W. Black and Isabel Trancoso (2015b). "Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 1520–1530 (see pages 28, 30, 71, 73, 78).

Thang Luong, Hieu Pham and Christopher D. Manning (2015). "Bilingual Word Representations with Monolingual Quality in Mind". In: *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*. Denver, Colorado: Association for Computational Linguistics, pp. 151–159 (see page 28).

Thang Luong, Richard Socher and Christopher Manning (2013). "Better Word Representations with Recursive Neural Networks for Morphology". In: *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 104–113 (see page 71).

Xuezhe Ma and Eduard Hovy (2017). "Neural Probabilistic Model for Non-projective MST Parsing". In: *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Taipei, Taiwan: Asian Federation of Natural Language Processing, pp. 59–69 (see page 69).

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig and Eduard Hovy (2018). "Stack-Pointer Networks for Dependency Parsing". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 1403–1414. DOI: `10.18653/v1/P18-1130` (see page 69).

David M. Magerman (1994). "Natural Language Parsing as Statistical Pattern Recognition". PhD thesis. Stanford University. arXiv: `cmp-lg/9405009 [cs.CL]` (see page 37).

Diego Marcheggiani and Ivan Titov (2017). "Encoding Sentences with Graph Convolutional Networks for Semantic Role Labeling". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 1506–1515. DOI: `10.18653/v1/D17-1159` (see page 133).

Mitchell P. Marcus, Mary Ann Marcinkiewicz and Beatrice Santorini (1993). "Building a Large Annotated Corpus of English: The Penn Treebank". In: *Computational Linguistics* 19.2, pp. 313–330 (see pages 72, 85, 175).

David McClosky, Eugene Charniak and Mark Johnson (2006). "Effective Self-Training for Parsing". In: *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*. New York City, USA: Association for Computational Linguistics, pp. 152–159 (see page 126).

Tom McCoy, Ellie Pavlick and Tal Linzen (2019). "Right for the Wrong Reasons: Diagnosing Syntactic Heuristics in Natural Language Inference". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, pp. 3428–3448. DOI: `10.18653/v1/P19-1334` (see page 2).

Warren S. McCulloch and Walter Pitts (1943). "A Logical Calculus of the Ideas Immanent in Nervous Activity". In: *The Bulletin of Mathematical Biophysics* 5.4, pp. 115–133 (see page 11).

Ryan McDonald, Koby Crammer and Fernando Pereira (2005a). "Online Large-Margin Training of Dependency Parsers". In: *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*. Ann Arbor, Michigan: Association for Computational Linguistics, pp. 91–98. DOI: `10.3115/1219840.1219852` (see pages 53, 56, 57).

Ryan McDonald, Kevin Lerman and Fernando Pereira (2006). "Multilingual Dependency Analysis with a Two-Stage Discriminative Parser". In: *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*. New York City, USA: Association for Computational Linguistics, pp. 216–220 (see pages 59, 82, 83).

Ryan McDonald and Joakim Nivre (2007). "Characterizing the Errors of Data-Driven Dependency Parsing Models". In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. Prague, Czech Republic: Association for Computational Linguistics, pp. 122–131 (see page 60).

Ryan McDonald and Fernando Pereira (2006). "Online Learning of Approximate Dependency Parsing Algorithms". In: *11th Conference of the European Chapter of the Association for Computational Linguistics* (see pages 57, 58, 60, 136, 148).

Ryan McDonald, Fernando Pereira, Kiril Ribarov and Jan Hajič (2005b). "Non-Projective Dependency Parsing using Spanning Tree Algorithms". In: *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*. Vancouver, British Columbia, Canada: Association for Computational Linguistics, pp. 523–530 (see pages 53, 56, 58).

Ryan McDonald and Giorgio Satta (2007). "On the Complexity of Non-Projective Data-Driven Dependency Parsing". In: *Proceedings of the Tenth International Conference on Parsing Technologies*. Prague, Czech Republic: Association for Computational Linguistics, pp. 121–132 (see pages 58, 179).

Igor' Aleksandrovič Mel'čuk (1988). *Dependency syntax: theory and practice*. SUNY press (see page 36).

Tomas Mikolov, Kai Chen, Greg Corrado and Jeffrey Dean (2013a). "Efficient Estimation of Word Representations in Vector Space". In: *Proceedings of the 1st International Conference on Learning Representations, Workshop Track*. Scottsdale, AZ, USA. arXiv: `1301.3781 [cs.CL]` (see pages 25, 26).

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado and Jeff Dean (2013b). "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems 26 (NIPS 2013)*. Ed. by C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani and K. Q. Weinberger. Vol. 26. Curran Associates, Inc., pp. 3111–3119 (see pages 25, 26, 103).

Thomas Mueller, Helmut Schmid and Hinrich Schütze (2013). "Efficient Higher-Order CRFs for Morphological Tagging". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, WA, USA: Association for Computational Linguistics, pp. 322–332 (see pages 109, 136).

Frank Henrik Müller (2004). "Stylebook for the Tübingen Partially Parsed Corpus of Written German (TüPP-D/Z)". In: *Sonderforschungsbereich 441, Seminar für Sprachwissenschaft, Universität Tübingen*. Vol. 28, p. 2006 (see page 103).

Jens Nilsson, Joakim Nivre and Johan Hall (2007). "Generalizing Tree Transformations for Inductive Dependency Parsing". In: *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*. Prague, Czech Republic: Association for Computational Linguistics, pp. 968–975 (see page 49).

Joakim Nivre (2003). "An Efficient Algorithm for Projective Dependency Parsing". In: *Proceedings of the Eighth International Conference on Parsing Technologies*. Nancy, France, pp. 149–160 (see pages 46, 52).

Joakim Nivre (2004). "Incrementality in Deterministic Dependency Parsing". In: *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*. Barcelona, Spain: Association for Computational Linguistics, pp. 50–57 (see page 41).

Joakim Nivre (2008). "Algorithms for Deterministic Incremental Dependency Parsing". In: *Computational Linguistics* 34.4, pp. 513–553. DOI: `10.1162/coli.07-056-R1-07-027` (see pages 41, 43).

Joakim Nivre (2009). "Non-Projective Dependency Parsing in Expected Linear Time". In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. Suntec, Singapore: Association for Computational Linguistics, pp. 351–359 (see page 49).

Joakim Nivre, Johan Hall and Jens Nilsson (2006a). "MaltParser: A Data-Driven Parser-Generator for Dependency Parsing". In: *Proceedings of the Fifth International*

Conference on Language Resources and Evaluation (LREC'06). Genoa, Italy: European Language Resources Association (ELRA) (see pages 60, 62).

Joakim Nivre, Johan Hall, Jens Nilsson, Gülşen Eryiğit and Svetoslav Marinov (2006b). "Labeled Pseudo-Projective Dependency Parsing with Support Vector Machines". In: *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*. New York City, USA: Association for Computational Linguistics, pp. 221–225 (see page 46).

Joakim Nivre, Marco Kuhlmann and Johan Hall (2009). "An Improved Oracle for Dependency Parsing with Online Reordering". In: *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*. Paris, France: Association for Computational Linguistics, pp. 73–76 (see page 49).

Joakim Nivre and Jens Nilsson (2005). "Pseudo-Projective Dependency Parsing". In: *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*. Ann Arbor, Michigan: Association for Computational Linguistics, pp. 99–106. DOI: `10.3115/1219840.1219853` (see page 48).

Marian Olteanu and Dan Moldovan (2005). "PP-Attachment Disambiguation Using Large Context". In: *Proceedings of the Human Language Technology Conference and the Conference on Empirical Methods in Natural Language Processing*. Vancouver, British Columbia, Canada: Association for Computational Linguistics, pp. 273–280 (see pages 94–98).

Patrick Pantel and Dekang Lin (2000). "An Unsupervised Approach to Prepositional Phrase Attachment using Contextually Similar Words". In: *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*. Hong Kong: Association for Computational Linguistics, pp. 101–108. DOI: `10.3115/1075218.1075232` (see pages 94, 96, 98).

Razvan Pascanu, Tomas Mikolov and Yoshua Bengio (2013). "On the Difficulty of Training Recurrent Neural Networks". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, GA, USA: JMLR Workshop and Conference Proceedings, pp. 1310–1318 (see page 17).

Jeffrey Pennington, Richard Socher and Christopher Manning (2014). "Glove: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. Doha, Qatar: Association for Computational Linguistics, pp. 1532–1543. DOI: `10.3115/v1/D14-1162` (see page 137).

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee and Luke Zettlemoyer (2018). "Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*

(Long Papers). New Orleans, Louisiana: Association for Computational Linguistics, pp. 2227–2237 (see pages 31, 32, 69).

Barbara Plank, Anders Søgaard and Yoav Goldberg (2016). "Multilingual Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Models and Auxiliary Loss". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 412–418 (see page 71).

Siyu Qiu, Qing Cui, Jiang Bian, Bin Gao and Tie-Yan Liu (2014). "Co-learning of Word Representations and Morpheme Representations". In: *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*. Dublin, Ireland: Dublin City University and Association for Computational Linguistics, pp. 141–150 (see page 28).

Adwait Ratnaparkhi (1998). "Statistical Models for Unsupervised Prepositional Phrase Attachment". In: *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics, Volume 2*. Montreal, Quebec, Canada: Association for Computational Linguistics, pp. 1079–1085. DOI: 10.3115/980691.980746 (see pages 93, 94, 98, 103).

Adwait Ratnaparkhi, Jeff Reynar and Salim Roukos (1994). "A Maximum Entropy Model for Prepositional Phrase Attachment". In: *Proceedings of the ARPA Workshop on Human Language Technology*. Plainsboro, New Jersey (see pages 91, 93, 94, 96, 98).

Vikas Raunak, Vivek Gupta and Florian Metze (2019). "Effective Dimensionality Reduction for Word Embeddings". In: *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*. Florence, Italy: Association for Computational Linguistics, pp. 235–243. DOI: 10.18653/v1/W19-4328 (see page 138).

Yoon-Hyung Roh, Ki-Young Lee and Young-Gil Kim (2011). "Improving PP Attachment Disambiguation in a Rule-based Parser". In: *Proceedings of the 25th Pacific Asia Conference on Language, Information and Computation*. Singapore: Institute of Digital Enhancement of Cognitive Processing, Waseda University, pp. 559–566 (see page 98).

Karin Kipper Schuler (2005). "Verbnet: A Broad-Coverage, Comprehensive Verb Lexicon". PhD thesis (see page 97).

Mike Schuster and Kuldip K. Paliwal (1997). "Bidirectional Recurrent Neural Networks". In: *IEEE Transactions on Signal Processing* 45.11, pp. 2673–2681. DOI: 10.1109/78.650093 (see page 66).

Djamé Seddah, Sandra Kübler and Reut Tsarfaty (2014). “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*. Dublin, Ireland: Dublin City University, pp. 103–109 (see pages 72, 85, 108, 136, 175–177).

Wolfgang Seeker, Ines Rehbein, Jonas Kuhn and Josef van Genabith (2010). “Hard Constraints for Grammatical Function Labelling”. In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Uppsala, Sweden: Association for Computational Linguistics, pp. 1087–1097 (see page 82).

Richard Socher, John Bauer, Christopher D. Manning and Andrew Y. Ng (2013). “Parsing with Compositional Vector Grammars”. In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 455–465 (see pages 18, 97, 126).

Richard Socher, Christopher D. Manning and Andrew Y. Ng (2010). “Learning Continuous Phrase Representations and Syntactic Parsing with Recursive Neural Networks”. In: *Proceedings of the NIPS 2010 Deep Learning and Unsupervised Feature Learning Workshop* (see pages 18, 97, 128).

Henning Sperr, Jan Niehues and Alex Waibel (2013). “Letter N-Gram-based Input Encoding for Continuous Space Language Models”. In: *Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 30–39 (see page 71).

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov (2014). “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15, pp. 1929–1958 (see pages 76, 86, 103).

Jiri Stetina and Makoto Nagao (1997). “Corpus Based PP Attachment Ambiguity Resolution with a Semantic Dictionary”. In: *Proceedings of the Fifth Workshop on Very Large Corpora* (see pages 96, 98).

Ilya Sutskever, Oriol Vinyals and Quoc V. Le (2014). “Sequence to Sequence Learning with Neural Networks”. In: *Advances in Neural Information Processing Systems 27 (NIPS 2014)*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence and K. Q. Weinberger. Vol. 27. Curran Associates, Inc., pp. 3104–3112 (see page 66).

Kai Sheng Tai, Richard Socher and Christopher D. Manning (2015). “Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks”. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*

(Volume 1: Long Papers). Beijing, China: Association for Computational Linguistics, pp. 1556–1566. DOI: 10.3115/v1/P15-1150 (see page 18).

Robert Endre Tarjan (1977). “Finding Optimum Branchings”. In: *Networks* 7.1, pp. 25–35 (see page 53).

Lucien Tesnière (1959). *Eléments de syntaxe structurale*. Librairie C. Klincksieck (see page 37).

Kristina Toutanova, Dan Klein, Christopher D. Manning and Yoram Singer (2003). “Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network”. In: *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 252–259 (see pages 85, 175).

Shyam Upadhyay, Manaal Faruqui, Chris Dyer and Dan Roth (2016). “Cross-Lingual Models of Word Embeddings: An Empirical Comparison”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 1661–1670 (see page 28).

Tim Van de Cruys (2011). “Two Multivariate Generalizations of Pointwise Mutual Information”. In: *Proceedings of the Workshop on Distributional Semantics and Compositionality*. Portland, OR, USA: Association for Computational Linguistics, pp. 16–20 (see page 103).

Clara Vania and Adam Lopez (2017). “From Characters to Words to in Between: Do We Capture Morphology?” In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 2016–2027. DOI: 10.18653/v1/P17-1184 (see pages 71, 79).

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser and Illia Polosukhin (2017). “Attention is All you Need”. In: *Advances in Neural Information Processing Systems 30 (NIPS 2017)*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett. Vol. 30. Curran Associates, Inc., pp. 5998–6008 (see pages 19, 20, 32, 120).

Oriol Vinyals, Ł ukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever and Geoffrey Hinton (2015). “Grammar as a Foreign Language”. In: *Advances in Neural Information Processing Systems 28 (NIPS 2015)*. Ed. by C. Cortes, N. Lawrence, D. Lee, M. Sugiyama and R. Garnett. Vol. 28. Curran Associates, Inc., pp. 2773–2781 (see pages 66, 67).

Martin Volk (2001). “Exploiting the WWW As a Corpus to Resolve PP Attachment Ambiguities”. In: *Proceedings of Corpus Linguistics*, pp. 601–606 (see pages 95, 98).

David Weiss, Chris Alberti, Michael Collins and Slav Petrov (2015). "Structured Training for Neural Network Transition-Based Parsing". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 323–333 (see pages 64, 68).

Marion Weller and Ulrich Heid (2012). "Analyzing and Aligning German compound nouns". In: *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*. Istanbul, Turkey: European Language Resources Association (ELRA), pp. 2395–2400 (see pages 73, 175).

Hiroyasu Yamada and Yuji Matsumoto (2003). "Statistical Dependency Analysis with Support Vector Machines". In: *Proceedings of the Eighth International Conference on Parsing Technologies*. Nancy, France, pp. 195–206 (see page 46).

Xingxing Zhang, Jianpeng Cheng and Mirella Lapata (2017). "Dependency Parsing as Head Selection". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Valencia, Spain: Association for Computational Linguistics, pp. 665–676 (see pages 67, 68, 72, 74–76, 82, 83, 85–88, 159).

Xingxing Zhang, Liang Lu and Mirella Lapata (2016). "Top-Down Tree Long Short-Term Memory Networks". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, pp. 310–320. DOI: `10.18653/v1/N16-1035` (see pages 18, 84).

Yu Zhang, Zhenghua Li and Min Zhang (2020). "Efficient Second-Order TreeCRF for Neural Dependency Parsing". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, pp. 3295–3305. DOI: `10.18653/v1/2020.acl-main.302` (see page 69).

Yue Zhang and Stephen Clark (2008). "A Tale of Two Parsers: Investigating and Combining Graph-based and Transition-based Dependency Parsing". In: *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*. Honolulu, Hawaii: Association for Computational Linguistics, pp. 562–571 (see pages 45, 48, 56, 61).

Yue Zhang and Joakim Nivre (2011). "Transition-based Dependency Parsing with Rich Non-local Features". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, OR, USA: Association for Computational Linguistics, pp. 188–193 (see pages 45, 48, 61, 181).

Yue Zhang and Joakim Nivre (2012). "Analyzing the Effect of Global Learning and Beam-Search on Transition-Based Dependency Parsing". In: *Proceedings of COLING 2012: Posters*. Mumbai, India: The COLING 2012 Organizing Committee, pp. 1391–1400 (see pages 47, 48).

Hao Zhou, Yue Zhang, Shujian Huang and Jiajun Chen (2015). "A Neural Probabilistic Structured-Prediction Model for Transition-Based Dependency Parsing". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 1213–1222. DOI: `10.3115/v1/P15-1117` (see page 64).

Hao Zhou, Yue Zhang, Shujian Huang, Junsheng Zhou, Xin-Yu Dai and Jiajun Chen (2016). "A Search-Based Dynamic Reranking Model for Dependency Parsing". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 1393–1402. DOI: `10.18653/v1/P16-1132` (see pages 125, 130).

Junru Zhou and Hai Zhao (2019). "Head-Driven Phrase Structure Grammar Parsing on Penn Treebank". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, pp. 2396–2408. DOI: `10.18653/v1/P19-1230` (see page 99).

Chenxi Zhu, Xipeng Qiu, Xinchi Chen and Xuanjing Huang (2015). "A Re-ranking Model for Dependency Parser with Recursive Convolutional Neural Network". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 1159–1168 (see pages 18, 125–127, 130, 134–137, 140, 141, 144).