Ensemble Dependency Parsing across Languages – Methodological Perspectives

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart zur Erlangung der Würde eines Doktors
der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Agnieszka Faleńska

aus Szczecin, Polen

Hauptberichter:	Prof. Dr. Jonas Kuhn
Mitberichter:	Prof. Dr. Dr. Joakim Nivre

Tag der mündlichen Prüfung: 28. September 2020

Institut für Maschinelle Sprachverarbeitung der Universität Stuttgart

2021

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed, and that I have marked any citations accordingly.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Stuttgart, July 1, 2021

Place, Date

Agnieszka Faleńska

Contents

Abstract **xix**

Überblick **xxi**

Acknowledgements **xxiii**

1 Introduction **1**
1.1 Summary of Research Gaps and Contributions 4
1.2 Outline 7
1.3 Publications 8

2 Background **13**
2.1 Dependency Grammar, Trees, and Treebanks 13
2.1.1 Dependency Trees 14
2.1.2 Dependency Treebanks 16
2.2 Dependency Parsing 19
2.2.1 The Standard NLP Pipeline Architecture 20
2.2.2 Transition-based Parsing 22
2.2.2.1 Transition Systems 23
2.2.2.2 Oracle Parsing 25
2.2.2.3 Transition Scoring and Feature Extraction 28
2.2.3 Graph-based Parsing 30
2.2.3.1 Tree Scoring and Feature Extraction 31
2.2.3.2 Graph-based Decoders 34
2.2.4 Neural Dependency Parsers 41
2.2.4.1 Neural Networks and Non-linear Functions 41
2.2.4.2 Distributed Word Representations 44
2.2.4.3 Sentential Context 47

2.3 Training Statistical Dependency Parsers . . . 52
2.4 Conclusion . . . 55

3 Ensemble Dependency Parsers – Methods and Applications 59
3.1 Ensemble Methods and Dependency Parsers . . . 60
3.1.1 Combination during Training: Feature-based Stacking . . . 60
3.1.2 Combination during Prediction: Blending . . . 63
3.1.3 Feature-based Stacking vs. Blending . . . 64
3.1.4 Other Ensemble Approaches . . . 66
3.2 Ensemble Dependency Parsers and Applications . . . 68
3.2.1 Combining Strengths of Parsing Architectures . . . 68
3.2.2 Improving Downstream Tasks . . . 71
3.2.3 Resource Creation . . . 73
3.3 Conclusion . . . 76

4 Supertagging vs. Stacking for Dependency Parsing 79
4.1 Experimental Setup . . . 82
4.1.1 Datasets and Preprocessing . . . 82
4.1.2 Supertag Design . . . 83
4.1.3 Notation and Evaluation . . . 84
4.1.4 Parsers and Feature Models . . . 85
4.2 Comparing Supertagging and Stacking . . . 88
4.2.1 Supertagging and Stacking Accuracy . . . 88
4.2.2 In-Depth Analysis . . . 89
4.2.3 Oracle Experiments . . . 91
4.2.4 Self-Application . . . 93
4.3 Supertagging without Parsers . . . 93
4.3.1 Comparison with Level 0 Parsers . . . 94
4.3.2 Out-of-Domain Application . . . 96
4.4 Combining Supertagging and Stacking . . . 98
4.5 Conclusion . . . 99

5 Cross-lingual vs. Monolingual Parsing in Low-Resource Scenarios 101
5.1 Experimental Setup . . . 105
5.1.1 Low-Resource Scenarios . . . 105
5.1.2 Cross-lingual Methods . . . 106
5.1.2.1 Similarity Measures . . . 106

5.1.2.2 Source Weights . . . 107
5.1.3 Datasets . . . 108
5.1.4 Tools and Evaluation . . . 110
5.2 Cross-lingual Techniques vs. Monolingual Parsers . . . 111
5.2.1 The EXTPOS Scenario . . . 111
5.2.2 The TBPOS Scenario . . . 114
5.2.3 Overall Picture . . . 115
5.3 Application to Test Languages . . . 116
5.4 Conclusion . . . 117

6 Integrating Dependency Parsers in the Deep Contextualized Era 121
6.1 Experimental Setup . . . 123
6.1.1 Parsing Model Architecture . . . 123
6.1.2 Integration Methods . . . 125
6.1.3 Datasets and Preprocessing . . . 127
6.1.4 Evaluation and Analysis . . . 127
6.2 Diversity-Based Integration . . . 128
6.3 Parsing Architectures and Diversity . . . 130
6.3.1 Feature-based Stacking . . . 130
6.3.2 Blending . . . 132
6.4 Representations and Treebank-Specific Diversity . . . 133
6.4.1 Representation Analysis . . . 133
6.4.2 BiLSTMs Integration . . . 135
6.5 Conclusion . . . 138

7 The Utility of Structural Features in BiLSTM-based Dependency Parsers 139
7.1 Experimental Setup . . . 140
7.1.1 Parsing Model Architecture . . . 141
7.1.2 Data Sets and Preprocessing . . . 143
7.1.3 Evaluation and Implementation Details . . . 144
7.2 Structural Features and BiLSTMs . . . 144
7.2.1 Simple vs. Extended Architectures . . . 144
7.2.2 Influence of BiLSTMs . . . 145
7.3 Implicit Structural Context . . . 148
7.3.1 Structure and BiLSTM Representations . . . 148
7.3.2 Structure and Information Flow . . . 150

7.3.3 Structure and Performance . . . 152
7.4 Conclusion . . . 154

8 Conclusion **157**
8.1 Contributions . . . 157
8.2 A Broader Perspective . . . 160
8.2.1 Understanding the Behavior of Statistical Models . . . 160
8.2.2 Balancing between the Complexity and Accuracy of Architectures 161

A Treebank Statistics **165**
A.1 SPMRL 2014 Shared Task Datasets . . . 165
A.2 Universal Dependencies ver. 2.0 . . . 166
A.3 Universal Dependencies ver. 2.4 . . . 168

B Hyperparameters and Additional Analysis **169**
B.1 Appendix for Chapter 2 . . . 170
B.2 Appendix for Chapter 6 . . . 171
B.3 Appendix for Chapter 7 . . . 174

Bibliography **177**

List of Figures

1.1 An example dependency tree 2

2.1 A projective dependency tree 15
2.2 A non-projective dependency tree 16
2.3 Parallel sentences annotated according to three different annotation schemata 17
2.4 Parallel sentences from Figure 2.3 annotated according to the Universal Dependencies schema 18
2.5 Schematic view of a typical NLP pipeline architecture 20
2.6 Result of the pipeline from Figure 2.5 for an example sentence *"John gave Mary some flowers."* 21
2.7 Dependency tree from Figure 2.2 annotated with projective order and maximal projective components 26
2.8 Oracle parsing result for the tree in Figure 2.7 and the lazy SWAP oracle . . 26
2.9 Example indicator functions 28
2.10 An example of second-order factors 32
2.11 First- and higher-order factors as presented by Zhang and Zhao (2015) . . 32
2.12 Illustration of the Chu-Liu-Edmonds algorithm on an example sentence *"The pudding fell"* 37
2.13 Types of structures built by Eisner's algorithm 39
2.14 Rules for combining structures in Eisner's algorithm 39
2.15 A feed-forward neural network with two hidden layers 42
2.16 Schematic illustration of Chen and Manning's (2014) architecture scoring transitions for an example configuration 45
2.17 Schematic illustration of Pei et al.'s (2015) first-order architecture scoring an arc 46
2.18 Graphical representation of a simple recurrent neural network 48

2.19 Schematic illustration of Kiperwasser and Goldberg's (2016b) architecture of BiLSTM-based neural dependency parsers 51

3.1 A general view on an ensemble dependency parsing architecture 59
3.2 A schematic view of the ensemble architectures studied in this dissertation 61
3.3 Three steps of blending for an example sentence 63
3.4 Development results of our submission to the CONLL 2017 shared task . 70

4.1 Supertags derived from a dependency tree 81
4.2 Setup for comparing stacking and supertagging 88
4.3 The accuracy of the graph-based and transition-based parser relative to sentence length 90
4.4 The dependency precision and recall relative to dependency length 91

5.1 Delexicalized cross-lingual parsing with three source languages and weights 105
5.2 Schematic overview of training data used in the experimental setups in Chapter 5 110
5.3 Average parsing accuracy and cutting point for the EXTPOS scenario . . . 112
5.4 Cutting points for all of the artificial low-resource languages in the EXTPOS scenario 113
5.5 Average parsing accuracy and cutting point in TBPOS scenario 115
5.6 Cutting points for all artificial target languages in TBPOS scenario 115

6.1 Schematic illustration of the integration methods used in Chapter 6 126
6.2 Average parsing accuracy for stacking with different types of Level 0 information 131
6.3 Average parsing accuracy for blending when models are trained with or without BiLSTMs and with or without ELMo 132
6.4 The average IMPACT of tokens on BiLSTM vectors trained with dependency parsers with respect to the token position 134
6.5 Dependency recall and precision relative to the head position on development sets 135

7.1 Schematic illustration of the K&G architecture of BiLSTM-based neural dependency parsers (adaptation of Figure 2.19) 142
7.2 Dependency recall and precision relative to arc length on development sets 146
7.3 Parsing accuracy with incremental extensions to the feature set of the transition-based parser 147

7.4 Parsing accuracy with incremental extensions to the feature set of the graph-based parser 147
7.5 The average impact of tokens on BiLSTM vectors trained with dependency parser with respect to the surface distance and the structural relation between them 149
7.6 Positions with the highest average impact on the MLP scores and their frequency for the transition-based parser (TbMin) 151
7.7 Positions with the highest impact on the MLP scores and their frequency for the graph-based parser (GbMin) 152
7.8 The performance drops of the transition-based parser (TbMin) when tokens at particular positions are removed from the BiLSTM encoding . . . 153
7.9 The performance drops of the graph-based parser (GbMin) when tokens at particular positions are removed from the BiLSTM encoding 154

B.1 Example of backtracking in Eisner's algorithm 170
B.2 Average LAS relative to sentence length on development sets (experimental setting described in Chapter 6) 173
B.3 The dependency recall and precision relative to arc length on development sets (experimental setting described in Chapter 6) 173
B.4 The dependency recall and precision relative to distance to root on development sets (experimental setting described in Chapter 6) 174

List of Tables

2.1 Transitions of the SWAPSTANDARD system (Nivre, 2009) 24
2.2 Example feature model from Zhang and Nivre (2011) 30
2.3 Exemplary feature model from McDonald et al. (2005a) for Eisner's decoder 33
2.4 Examples of non-linear functions used in neural networks 43

3.1 Additional stacking features from Nivre and McDonald (2008) 62

4.1 Morphological analyzers used for preprocessing in Chapter 4 84
4.2 Feature templates used for stacking and supertagging in the transition-based parser 86
4.3 Feature templates used for stacking and supertagging in the graph-based parser 87
4.4 Parsing results on test sets from the SPMRL 2014 shared task 89
4.5 Results on development sets for stacking and supertagging with different Level 0 information sources and Level 1 graph-based parser 92
4.6 Results on development sets for stacking and supertagging with different sources of Level 0 information and Level 1 transition-based parser 92
4.7 Supertag tagging accuracy on development sets 94
4.8 Results on development sets with a sequence labeler and a greedy transition-based parser as Level 0 tools 95
4.9 Stacking and supertagging results on the English Web Treebank for three different Level 1 parsers 97
4.10 Results on development sets for combining supertags and stacking 98

5.1 Summary of the two low-resource scenarios addressed in experiments in Chapter 5 106
5.2 Number of tokens and sentences in low-resource test-case treebanks . . . 109

5.3 Parsing results on the test, small and surprise languages 118

6.1 Average parsing results on test sets for stacking 128
6.2 Parsing results on test sets for blending 129
6.3 Average parsing results on test sets for multi-task learning 137

7.1 Average parsing results on test sets for simple and extended architectures 145

A.1 References for SPMRL 2014 shared task treebanks 165
A.2 Statistics for SPMRL 2014 shared task treebanks 166
A.3 Statistics for the treebanks from Universal Dependencies v. 2.0 that have development sets 168
A.4 A selection of treebanks from Universal Dependencies v. 2.4 used for experiments in Chapter 6 168

B.1 Hyperparameters for the parsers used in experimentation in Chapter 6 . . 171
B.2 Standard deviation for the results in Table 6.1 172
B.3 Standard deviation for the results in Table 6.3 172
B.4 Hyperparameters for the parsers used in experiments in Chapter 7 174
B.5 Standard deviation for the results in Table 7.1 175

List of Algorithms

2.1 Greedy transition-based parsing with a statistical model 22
2.2 The lazy SWAP oracle from Nivre et al. (2009) 25
2.3 Graph-based parsing with a statistical model 31
2.4 The Chu-Liu-Edmonds algorithm for finding the maximum spanning tree in a given graph 35
2.5 Part of the Chu-Liu-Edmonds algorithm responsible for contracting cycles into single nodes 36
2.6 Part of the Chu-Liu-Edmonds algorithm responsible for resolving contracted cycles 36
2.7 Eisner's algorithm for finding the maximum spanning projective tree for a given sentence 39
2.8 General online learning framework 53

List of Abbreviations

BiLSTM	Bi-Directional Long Short-Term Memory
CCG	Combinatory Categorial Grammar
CKY	Cocke-Kasami-Younger
CLE	Chu-Liu-Edmonds
CRF	Conditional Random Fields
HMM	Hidden Markov Models
HPSG	Head-Driven Phrase Structure Grammar
LAS	Labeled Attachment Score
LFG	Lexical Functional Grammar
LSTM	Long Short-Term Memory
LTAG	Lexicalized Tree Adjoining Grammar
MLP	Multi-layer Perceptron
MST	Maximum Spanning Tree
MTL	Multi-task Learning
NLP	Natural Language Processing
POS	Part-of-Speech
PTB	Penn Treebank
RNN	Recurrent Neural Network

SRNN	Simple Recurrent Neural Network
SVMs	Support Vector Machines
UD	Universal Dependencies
UPOS	Universal Part-of-Speech
WALS	World Atlas of Language Structures

Abstract

Human language is ambiguous. Such ambiguity occurs at the lexical as well as syntactic level. At the lexical level, the same word can represent different concepts and objects. At the syntactic level, one phrase or a sentence can have more than one interpretation. Language ambiguity is one of the biggest challenges of **Natural Language Processing (NLP)**, i.e., the research field that sits at the intersection of machine learning and linguistics, and that deals with automatic processing of language data. This challenge arises when automatic NLP tools need to resolve ambiguities and select one possible interpretation of a text to approach understanding its meaning.

This dissertation focuses on one of the essential Natural Language Processing tasks – **dependency parsing**. The task involves assigning a syntactic structure called a dependency tree to a given sentence. Parsing is usually one of the processing steps that helps downstream NLP tasks by resolving some of the syntactic ambiguities occurring in sentences. Since human language is highly ambiguous, deciding on the best syntactic structure for a given sentence is challenging. As a result, even state-of-the-art dependency parsers are far from being perfect.

Ensemble methods allow for postponing the decision about the best interpretation until several single parsing models express their opinions. Such complementary views on the same problem show which parts of the sentence are the most ambiguous and require more attention. Ensemble parsers find a consensus among such single predictions, and as a result, provide robust and more trustworthy results.

Ensemble parsing architectures are commonly regarded as solutions only for experts and overlooked in practical applications. Therefore, this dissertation aims to provide a deeper understanding of ensemble dependency parsers and answer practical questions that arise when designing such approaches. We investigate ensemble models from three

core **methodological perspectives**: parsing time, availability of training resources, and the final accuracy of the system. We demonstrate that in applications where the complexity of the architecture is not a bottleneck, an integration of strong and diverse parsers is the most reliable approach. Such integration provides robust results regardless of the language and the domain of application. However, when the final accuracy of the system can be sacrificed, more efficient ensemble architectures become available. The decision on how to design them has to take into consideration the desired parsing time, the available training data, and the involved single predictors.

The main goal of this thesis is to investigate ensemble parsers. However, to design an ensemble architecture for a particular application, it is crucial to understand the similarities and differences in the behavior of its components. Therefore, this dissertation makes contributions of two sorts: (1) we provide guidelines on practical applications of ensemble dependency parsers, but also (2) through the ensembles, we develop a deeper understanding of single parsing models. We primarily focus on differences between the traditional parsers and their recent successors, which use deep learning techniques.

Überblick

Die menschliche Sprache ist mehrdeutig. Diese Mehrdeutigkeit tritt sowohl auf lexikalischer als auch auf syntaktischer Ebene auf. Auf lexikalischer Ebene kann ein und dasselbe Wort verschiedene Konzepte und Objekte repräsentieren. Auf syntaktischer Ebene kann eine Phrase oder ein Satz mehr als eine Interpretation haben. Sprachliche Mehrdeutigkeit ist eine der größten Herausforderungen der **Maschinellen Sprachverarbeitung (NLP)**, des Bereichs des maschinellen Lernens, der sich mit sprachlichen Daten befasst. Diese Herausforderung entsteht, wenn automatische NLP-Anwendungen Mehrdeutigkeiten auflösen und eine mögliche Interpretation eines Textes auswählen müssen, um sich dem Verständnis seiner Bedeutung anzunähern.

Diese Dissertation konzentriert sich auf eine der wesentlichen Aufgaben der Natürlichen Sprachverarbeitung – das **Dependenz-Parsing**. Die Aufgabe besteht darin, einem Satz eine syntaktische Struktur, einen sogenannten Dependenzbaum, zuzuweisen. Parsing ist normalerweise einer der Verarbeitungsschritte, der den nachfolgenden NLP-Anwendungen hilft, indem er einige der syntaktischen Mehrdeutigkeiten, die in Sätzen auftreten, auflöst. Da die menschliche Sprache hochgradig mehrdeutig ist, ist die Entscheidung über die beste syntaktische Struktur eines gegebenen Satzes eine Herausforderung. Infolgedessen sind selbst hochmoderne Dependenz-Parser weit davon entfernt, perfekt zu sein.

Ensemble-Methoden erlauben es, die Entscheidung über die beste Interpretation aufzuschieben, bis mehrere Experten, d.h. einzelne Parsing-Modelle, ihre Meinung äußern. Solche sich ergänzenden Ansichten zum gleichen Problem zeigen, welche Teile des Satzes am mehrdeutigsten sind und damit mehr Aufmerksamkeit benötigen. Ensemble-Parser finden einen Konsens unter solchen Einzelvorhersagen und liefern als Ergebnis robuste und vertrauenswürdigere Ergebnisse.

Ensemble-Parsing-Architekturen werden gemeinhin nur als Lösungen für Experten angesehen und in der praktischen Anwendung übersehen. Daher zielt diese Dissertation darauf ab, ein tieferes Verständnis von Ensemble-Dependenz-Parsern zu vermitteln und praktische Fragen zu beantworten, die beim Entwurf solcher Ansätze auftreten. Wir untersuchen Ensemble-Modelle aus drei **methodischen Kernperspektiven**: Parsingzeit, Verfügbarkeit von Trainingsressourcen und die resultierende Genauigkeit des Systems. Wir zeigen, dass in Anwendungen, bei denen die Komplexität der Architektur keine Rolle spielt, eine Integration von starken und unterschiedlichen Parsern der zuverlässigste Ansatz ist. Eine solche Integration liefert unabhängig von der Sprache und der Anwendungsdomäne robuste Ergebnisse. Wenn andererseits Abstriche bei der Genauigkeit des Systems gemacht werden können, werden effizientere Ensemble-Architekture möglich. Bei der Entscheidung, wie diese zu gestalten sind, müssen die gewünschte Parsingzeit, die verfügbaren Trainingsdaten und die beteiligten Einzelprädiktoren berücksichtigt werden.

Das Hauptziel dieser Arbeit ist die Untersuchung von Ensemble-Parsern. Um jedoch eine Ensemble-Architektur für eine bestimmte Anwendung zu entwerfen, ist es entscheidend, die Ähnlichkeiten und Unterschiede im Verhalten ihrer Komponenten zu verstehen. Daher werden in dieser Dissertation zwei Arten von Beiträgen geleistet: (1) Wir geben Richtlinien für die praktische Anwendung von Ensemble-Dependenz-Parsern, sowie (2) Durch die Ensembles entwickeln wir ein tieferes Verständnis der einzelnen Parsing-Modelle. Wir konzentrieren uns in erster Linie auf die Unterschiede zwischen den traditionellen Parsern und ihren jüngsten Nachfolgern, die Techniken des Deep-Learning anwenden.

Acknowledgements

First and foremost, I am grateful to my supervisor Jonas Kuhn. Thank you for your support and encouragement throughout this whole process. Witnessing your enthusiasm to explore new research topics and questions along with your impressive ability to see each of them within a broader perspective is always very inspiring. Not to mention your ensemble saxophone skills!

I would like to express my gratitude to Joakim Nivre, for accepting to be my external reviewer, and my doctoral committee members Michael Sedlmair and Thomas Ertl, for the stimulating discussion during my PhD defense.

My adventure with NLP would not start without Paweł Rychlikowski. Paweł not only introduced me to the topic of language processing but also infected me with his passion for it. Thank you for showing me that NLP can be not only interesting but also simply fun. I believe that the experience of working at Neurosoft was one of the crucial lessons that allowed me to be successful at my doctoral studies. I have to thank Cezary Dołęga for making it possible and giving me challenging responsibilities as well as the freedom to learn along the way.

I joined IMS because of two people – Anders Björkelund and Wolfgang Seeker. Thank you for answering that first odd email, welcoming me to Stuttgart, and showing me how to survive my PhD. More importantly, I have learned from you what type of research I want to pursue – the one that is not about the numbers but about digging deeper until you understand what hides underneath.

Special thanks go to my friends and colleagues, with whom I had the pleasure to work at IMS. Especially to my two co-authors – Özlem Çetinoğlu and Xiang Yu – for endless discussions, valuable feedback, and motivating me to look past the task of dependency

parsing. Together with Diego Frassinelli, Gabriella Lapesa, Dominik Schlechtweg, Enrica Troiano, and Sabine Schulte im Walde, you were my support group during the completion of this dissertation. Thank you all for cheering me up during this most challenging part, but also for having a lot of fun along the way. Moreover, thanks again to Özlem, Gabriella, Enrica, Xiang, Simon Tannert, and Paweł Laskoś-Grabowski for proofreading portions of the thesis.

Finally, I would like to thank my family for always being there when I needed them. If not for Michał and Joanna this thesis would not exist. Thank you for believing in me and giving me the strength and freedom to follow this path.

Chapter 1

Introduction

Human language is ambiguous. Such ambiguities occur at **the lexical level**, where the same word can represent different concepts and objects, and at **the syntactic level**, where a sentence can have more than one interpretation. Yet, human interactions rarely lead to misunderstandings. One of the reasons is that people are equipped with complex cognitive skills and world knowledge that help them understand the intended meaning of the linguistic input they are exposed to.

With this in mind, it is no surprise that language ambiguity is one of the biggest challenges in **Natural Language Processing (NLP)**, i.e., the research field that sits at the intersection of machine learning and linguistics, and that deals with automatic processing of language data (Goldberg, 2017, p. 1). NLP involves designing tools such as dialog systems or methods for automatic translation, and indeed, one of the crucial processing steps that these tools need to perform is resolving language ambiguities. For example, to answer the question "*Where is the nearest bank?*" a dialog system must decide which of the possible interpretations of the word *bank* the user referred to.[1] Moreover, automatic tools take into consideration interpretations that humans might not even be aware of. To illustrate this phenomenon, consider this passage from the book "*Harry Potter and the Chamber of Secrets*" (Rowling, 2000, Chapter 2, p. 19):

(1) The pudding fell to the floor with a heart-stopping crash. Cream splattered the windows and walls as the dish shattered. With a crack like a whip, Dobby vanished.

For humans, this text is rather clear and answering questions such as "*How did the pudding fall?*" or "*Was there any noise?*" should not cause any trouble. In contrast, NLP tools

[1] The word *bank* can mean a financial institution as well as a riverbank.

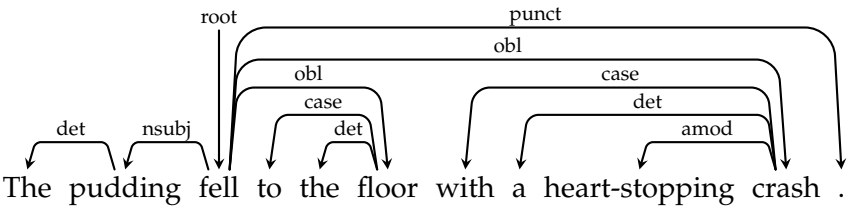


Figure 1.1: A dependency tree for the first sentence from example (1).

encounter difficulties already with the first sentence of this passage. For example, the well-known automatic translation system DeepL[2] translates it into an inaccurate Polish sentence "*Pudding spadł na podłogę w wyniku uderzenia serca.*" (eng. The pudding fell to the floor as a result of (one) beat of a heart.). Similarly, the German translation "*Der Pudding fiel mit einem Herzstillstand auf den Boden.*" (eng. The pudding fell to the floor in cardiac arrest.) is far from perfect.[3]

The challenges in automatic processing of the first sentence from example (1) stem from ambiguities that humans might not even notice. For example, at the lexical level, the word *crash* can describe a loud noise or an accident. At the syntactic level, *stopping* can be either a separate token that modifies the noun *heart* or a part of the adjectival participle *heart-stopping*, which in turn describes a property of the word *crash*. Although it is difficult to pinpoint what exactly went wrong during DeepL's translation, our hypothesis is that the error occurred already in the early stages of processing, where the noun *heart* was identified. Moreover, the combination of the word *heart* and the word *crash* interpreted as a synonym of an accident might have led to the *cardiac arrest* in the German translation.

In order to limit the possible ambiguities that occur in a sentence, automatic tools first assign a syntactic structure to the given sentence and then use this structure in subsequent processing steps (Green and Žabokrtský, 2016; Huang and Carley, 2019, among others). This dissertation focuses on the NLP task called **dependency parsing**, which given a sentence, produces the underlying syntactic structure in the form of a dependency graph.

Figure 1.1 presents a dependency structure, more specifically a dependency tree, for the first sentence from example (1). The tree encodes information that the verb *fell* is the root predicate of this sentence, and *(The) pudding* is its subject. Moreover, both *(to the)*

[2] https://www.deepl.com/translator
[3] Both Polish and German translations are results of DeepL on 28.06.2020.

floor and *(with a heart-stopping) crash* are dependents of the verb and not modifiers of the noun *pudding*. Finally, the participle *heart-stopping* describes a property of the word *crash*.

Data-driven techniques have proven highly effective in solving tasks that replicate complex cognitive skills, such as finding the syntactic structure underlying a word sequence. In this dissertation, we focus on **data-driven and supervised methods** for dependency parsing. Although unsupervised approaches to parsing have been proposed (Klein and Manning, 2004), their performance is far behind the supervised methods.

Data-driven supervised parsers are guided by **statistical classifiers**. These machine learning methods require relatively large data collections, labeled with the correct target representations. This data is used for supervised training, i.e., adjusting the parameters of the classifiers to learn and generalize the training structures. The quality of classifiers directly translates to the quality of parsers. Although such classifiers are able to capture even subtle cross-level effects that drive human language understanding, like other statistical tools, they are not infallible. Their mistakes might be caused by differences in the training and target data domains, unknown words, or unusual expressions. Moreover, these mistakes cause the parsers to predict ungrammatical structures or select syntactically sound but incorrect interpretations for ambiguous sentences. Such parsing errors are difficult to detect and cause problems in the downstream applications, such as the imprecise translation that we saw earlier.

Ensemble methods are a common approach to deal with classification errors (Dietterich, 2000). The approach follows an intuition given by Polikar (2006) – in real-life, to make a well-informed financial, medical, or social decision, it is important to consult several experts and weight their individual opinions to reach the final decision. Ensemble methods apply this intuition to machine learning. They approach a specific task with a number of distinct classifiers and try to find a general agreement among their predictions.

Ensemble methods have a long tradition in dependency parsing. The reason is that they postpone the difficult decision about the best syntactic interpretation of a sentence until several experts, i.e., single parsing models, express their opinions. Such complementary views on the same problem show which parts of the sentence are the most ambiguous and require particular attention. **Ensemble dependency parsers** find a consensus among such individual predictions and, as a result, provide robust and more trustworthy results. Across parsing literature, ensembles proved their effectiveness in exploiting differences in parsing algorithms (Nivre and McDonald, 2008; Sagae and Lavie, 2006) as well as combining predictions of parsers trained on data from different languages

(Rosa and Žabokrtský, 2015). Moreover, they were extensively promoted by a series of shared tasks focusing on multilingual parsing (Buchholz and Marsi, 2006; Zeman et al., 2018, among others).

Even if ensembles have been proved extremely useful in mitigating parsing errors, research publications about practical, especially large-scale applications of ensemble parsers, are surprisingly rare. One of the possible reasons might be that since ensembles involve multiple parsers, they are more complex than single models and, as a result, regarded as solutions "*unusable by all but experts*" (Choi et al., 2015). Moreover, when it comes to practical applications, a variety of **methodological aspects** have to be considered already when designing a parsing architecture consisting of a single parser (Dakota and Kübler, 2018). Such aspects involve differences in off-the-shelf parsing tools, the specificity of the domain of application, and the interrelations between the selected parsing algorithm and the particularities of the language to parse. The picture for the ensemble dependency parsers is even more complicated. First of all, ensembles inherit the challenges associated with the individual models they are built from. Secondly, since the ensembles repeatedly proved their effectiveness, they are commonly assumed beneficial regardless of the use case. Such an assumption leads to applications that report the performance of the ensemble but not the single models, making it impossible to determine if the ensemble component was indeed profitable in a particular application (e.g., Schweitzer et al. (2018); Shi et al. (2017b)). Finally, ensembles come with higher processing cost than single models, and as a result, they are commonly regarded as inadequate for speed-sensitive application (Kuncoro et al., 2016).

In this thesis, we perform a systematic analysis of ensemble dependency parsers and address the methodological aspects related to their construction and application. Our goal is to provide a better understanding of these methods, which is crucial from the perspective of the parsing community as well as practical downstream applications.

1.1 Summary of Research Gaps and Contributions

Three essential questions have to be addressed when designing an ensemble parsing system: "*How fast does my system have to be?*", "*How much training data do I have?*", and "*How to achieve the best possible accuracy for my application?*". Out of these questions, we extract the three methodological perspectives that constitute the core of this dissertation: parsing time, availability of training data, and the final accuracy of the system. We first give

an overview of these perspectives and summarize the related research gaps and our contributions. Then, we describe how we address these gaps throughout the chapters of this thesis.

Parsing time. Parsing is commonly considered a task with a relatively high computational cost (Gómez-Rodríguez et al., 2017). This processing cost varies across different parsing algorithms and is usually directly related to their accuracy, i.e., the more accurate the model, the higher the computational cost. Ensemble methods are high up in this spectrum because they improve the accuracy by requiring to run multiple parsing algorithms at application time. Due to this additional overhead, they are not the preferred choice in applications where speed is essential, such as parsing large-scale web data (Clark et al., 2009). In this context, there is *a clear need for lighter and faster ensemble systems*.

A straightforward approach to design a faster ensemble architecture is to use faster individual models. We push this idea forward and investigate *whether the involved models have to predict well-formed dependency trees.* Specifically, we compare two architectures. The first one is a typical ensemble built from two dependency parsers. In the second, we replace one of these parsers with a faster tool called *supertagger*. Supertagging, also referred to as *almost parsing* (Joshi and Bangalore, 1994), operates on tags that are similar to Part-of-Speech (POS) tags, but encode shallow syntactic information. Therefore, the method does not require parsers and can be performed with fast taggers. We find that although such supertags do not convey as much information as full trees, they can in practice improve a dependency parser to an equal extent. Therefore, when speed is essential, supertagging allows for the design of ensemble architectures with faster individual components.

Availability of training data. Statistical dependency parsers require manually annotated data, i.e., gold-standard treebanks, to learn from. However, such treebanks exist only for about 1% of languages spoken in the world (Agić, 2017). Therefore, when designing a parsing system for a language outside of this group, it is essential to decide between two options: either (1) dedicating resources to build a new treebank to train a *monolingual* parser, or (2) pursuing *cross-lingual* techniques that take advantage of resources available for other languages. Both of these options come with their own challenges: creating treebanks is costly (Zeman and Resnik, 2008; Souček et al., 2013), and cross-lingual methods strongly depend on the availability of other manually-annotated resources, such as POS-annotated corpora. Yet, *clear guidelines concerning the circumstances under which monolingual parsers should be preferred over cross-lingual techniques are missing*.

We aim at developing such guidelines with a strong focus on highly low-resource languages, for which only a few manually annotated trees exist. The central question we examine is *whether we can find cases where cross-lingual parsers achieve stronger accuracy than monolingual ones*. Specifically, we look at ensemble cross-lingual techniques that transfer structural information from several source languages to the low-resource target. We demonstrate that two critical factors have to be considered when deciding between ensemble cross-lingual and monolingual models: the quality of POS taggers for the target language itself and the availability of treebanks for typologically similar languages. Cross-lingual techniques are preferable when there are source languages close to the target, and high-quality POS tagging can be achieved. Otherwise, monolingual parsers can outperform cross-lingual ones even when trained only on few gold-standard sentences.

Accuracy of the system. Dependency parsers are rarely used on their own. More commonly, they provide input for subsequent NLP tasks, e.g., sentiment classification (Huang and Carley, 2019) or semantic role labeling (Wang et al., 2019). Since errors propagate in such sequential processing, the better the accuracy of the parsers, the better the performance of the downstream tasks. Therefore, end users commonly favor high-performing dependency parsers.

In applications where accuracy has priority, modern neural parsers are preferable. Such parsing architectures extended with deep learning techniques outperform traditional pre-neural parsing methods and have significantly advanced the state-of-the-art over the last few years. However, as stated by Manning (2015), developing new methods does not always go together with a deep understanding of how they work or will behave in new scenarios. The same applies to neural dependency parsers. They have been repeatedly proven highly effective, but now is the moment to invest research effort in understanding *how deep learning techniques translate into their impressive accuracy* and *if their performance can be further improved through ensemble methods*.

To address these two questions, we focus on particular deep learning enhancements – techniques that incorporate the context of the whole sentence into word representations. Architectures that depend on such sentential context have been shown very effective across many NLP tasks (Plank et al., 2016; Kiperwasser and Goldberg, 2016b, among others). We demonstrate that the accuracy of dependency parsers that include such rich word representations can be improved through ensemble methods. However, to obtain such improvements, the ensemble architecture has to be designed in a different way than for classical parsers. Traditionally, ensembles have been mostly known for their efficacy in exploiting differences in the two major parsing paradigms: transition-based and

graph-based. We show that with the employment of the modern word representations, these two architectures and their error distributions converge. Therefore, the most significant gains in accuracy can be obtained by combining multiple strong graph-based models and not parsers from different paradigms.

1.2 Outline

The remainder of this dissertation is organized as follows:

Chapter 2 provides some general background. We introduce the main concepts used in this thesis, such as dependency grammar, trees, and treebanks. We explain the two main parsing paradigms: transition-based and graph-based. Next, we describe how, over the last few years, traditional dependency parsers were successively extended with deep learning techniques. In particular, we focus on the neural techniques that will be crucial in the subsequent chapters, such as Bi-Directional Long Short-Term Memory (BiLSTM) and deep contextualized embeddings. Finally, we describe the online training framework used to train the statistical models throughout the dissertation.

Chapter 3 focuses on ensemble dependency parsers. First, we discuss the two ensemble methods used in this thesis: feature-based stacking and blending. Next, we provide an overview of other approaches for integrating dependency parsers and outline how they differ from stacking and blending. Finally, we turn to practical applications for ensemble dependency parsers and demonstrate a selection of possible use cases for ensemble techniques in parsing.

In **Chapter 4**, we look at ensemble dependency parsers from the first methodological perspective – **parsing time**. We address the question of how to design lighter and faster ensemble systems. We compare two methods for providing syntactic features for statistical dependency parsing – supertagging and feature-based stacking. The goal of this comparison is to find if the former method can replace the latter in applications where reducing processing time is crucial, i.e., parsing web data.

In **Chapter 5**, we turn to the second methodological perspective – **availability of training data**. Our focus in this chapter is on low-resource scenarios, for which few manually annotated trees exist in the language of interest. We compare the main two approaches that can be applied in such scenarios: (1) an ensemble cross-lingual technique called multi-source delexicalized parsing, and (2) monolingual parsers trained on small treebanks. We develop a methodology to choose one method over another depending on the

typological features of the language and the availability of corpora annotated with POS tags.

In **Chapter 6**, we move to the third and final methodological perspective – **the accuracy of the system**. For this purpose, we turn to the state-of-the-art neural dependency parsers. Specifically, we focus on architectures that employ BiLSTMs and deep contextualized representations. We investigate under what circumstances the performance of such architectures can be further improved through such ensemble methods as feature-based stacking and blending.

In **Chapter 7**, we stay within the context of **the accuracy of the system**. Taking into consideration the results from the previous chapter, namely that predictions of neural parsers lack diversity, we investigate changes that neural techniques brought into parsing architectures. Specifically, rich feature sets used to be the advantage of transition-based parsers over graph-based models. Since neural models do not use them, we investigate whether re-introducing such features would restore the diversity between these two paradigms, thus providing fruitful grounds for the use of ensembles.

Chapter 8 concludes this thesis. We summarize our findings on the evaluation of ensemble parsers in general and specifically with respect to the three methodological perspectives at the core of this dissertation: parsing time, availability of training data, and parsing accuracy. Next, we review the contributions of this dissertation from a broader perspective and discuss two themes that run throughout this thesis: understanding the behavior of statistical models and balancing between architectural complexity and accuracy. Finally, we discuss possible future developments for ensemble dependency parsers.

1.3 Publications

This thesis is based on the following publications:

- Faleńska, A., Björkelund, A., Çetinoğlu, Ö., and Seeker, W. (2015). Stacking or Supertagging for Dependency Parsing – What's the Difference? In *Proceedings of the 14th International Conference on Parsing Technologies*, pages 118–129, Bilbao, Spain. Association for Computational Linguistics

  Chapter 4 is based on this publication. My co-authors took part in discussing the underlying ideas and drafting the paper. The formulation and design of the experimental setup and running the experiments was my own contribution.

- Falenska, A. and Çetinoğlu, Ö. (2017). Lexicalized vs. Delexicalized Parsing in Low-Resource Scenarios. In *Proceedings of the 15th International Conference on Parsing Technologies*, pages 18–24, Pisa, Italy. Association for Computational Linguistics

  Chapter 5 is based on this publication. My co-author Özlem Çetinoğlu took part in discussing the underlying ideas and drafting the paper. The formulation and design of the experimental setup and running the experiments was my own contribution.

- Falenska, A. and Kuhn, J. (2019). The (Non-)Utility of Structural Features in BiLSTM-based Dependency Parsers. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 117–128, Florence, Italy. Association for Computational Linguistics

  Chapter 7 is based on this publication. The underlying ideas, design of the experimental setup, and running the experiments was my own contribution. My co-author Jonas Kuhn had an advisory role as I carried out this work and provided feedback on the final draft of the paper.

- Falenska, A., Björkelund, A., and Kuhn, J. (2020a). Integrating Graph-Based and Transition-Based Dependency Parsers in the Deep Contextualized Era. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 25–39, Online. Association for Computational Linguistics

  Chapter 6 is based on this publication. The underlying ideas, design of the experimental setup, and running the experiments was my own contribution. My co-authors Jonas Kuhn and Anders Björkelund had an advisory role and provided feedback on the drafts of the paper.

The final form of this dissertation is also a result of other peer-reviewed articles published in the course of my doctoral studies. Although they are not its core, the dissertation refers to them in relation to some less central concepts:

- Björkelund, A., Çetinoğlu, Ö., Faleńska, A., Farkas, R., Mueller, T., Seeker, W., and Szántó, Z. (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, pages 97–102, Dublin, Ireland. Dublin City University

- Björkelund, A., Falenska, A., Yu, X., and Kuhn, J. (2017). IMS at the CoNLL 2017 UD Shared Task: CRFs and Perceptrons Meet Neural Networks. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 40–51, Vancouver, Canada. Association for Computational Linguistics

- Falenska, A., Björkelund, A., Yu, X., and Kuhn, J. (2018). IMS at the PolEval 2018: A Bulky Ensemble Dependency Parser meets 12 Simple Rules for Predicting Enhanced Dependencies in Polish. In Ogrodniczuk, M. and Łukasz Kobyliński, editors, *Proceedings of the PolEval 2018 Workshop*, pages 25–39, Warsaw, Poland. Institute of Computer Science, Polish Academy of Sciences

- Schweitzer, K., Eckart, K., Gärtner, M., Falenska, A., Riester, A., Rösiger, I., Schweitzer, A., Stehwien, S., and Kuhn, J. (2018). German Radio Interviews: The GRAIN Release of the SFB732 Silver Standard Collection. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation*, pages 2887–2895, Miyazaki, Japan. European Language Resources Association

- Yu, X., Falenska, A., and Vu, N. T. (2017). A General-Purpose Tagger with Convolutional Neural Networks. In *Proceedings of the First Workshop on Subword and Character Level Models in NLP*, pages 124–129, Copenhagen, Denmark. Association for Computational Linguistics

- Yu, X., Falenska, A., and Kuhn, J. (2019b). Dependency Length Minimization vs. Word Order Constraints: An Empirical Study On 55 Treebanks. In *Proceedings of the First Workshop on Quantitative Syntax*, pages 89–97, Paris, France. Association for Computational Linguistics

- Yu, X., Falenska, A., Vu, N. T., and Kuhn, J. (2019c). Head-First Linearization with Tree-Structured Representation. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 279–289, Tokyo, Japan. Association for Computational Linguistics

- Yu, X., Falenska, A., Haid, M., Vu, N. T., and Kuhn, J. (2019a). IMSurReal: IMS at the Surface Realization Shared Task 2019. In *Proceedings of the 2nd Workshop on Multilingual Surface Realisation*, pages 50–58, Hong Kong, China. Association for Computational Linguistics

- Falenska, A., Czesznak, Z., Jung, K., Völkel, M., Seeker, W., and Kuhn, J. (2020b). GRAIN-S: Manually Annotated Syntax for German Interviews. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 5169–5177, Marseille, France. European Language Resources Association

Chapter 2

Background

This chapter provides the general background for the research presented in this dissertation. We start by introducing the main linguistic concepts that lay the foundations for dependency parsing, such as dependency grammar, trees, and treebanks. Primarily, we focus on Universal Dependencies, which give grounds for our cross-lingual studies in Chapter 5. Next, we turn to the methods for automatic prediction of dependency trees and explain the two main parsing paradigms – transition-based and graph-based. Then, we discuss how these two architectures were recently extended with deep learning methods. In particular, we focus on the BiLSTM-based dependency parsers, that are used in Chapters 6 and 7. Finally, we review the online training framework and methods used for training statistical models throughout this thesis.

2.1 Dependency Grammar, Trees, and Treebanks

Dependency grammar is a formalism that represents syntactic structures of sentences in terms of directed binary *dependency relations*. Such relations hold between two linguistic units: the head (also referred to as the governor or the parent) and the dependent (also called the modifier or the child). The dependency relations are motivated by the functions of linguistic units in a sentence. Depending on the grammar theory, such functions can be syntactic, e.g., subject and object, or semantic, e.g., agent and patient.

The dependency concept has a long tradition that can be traced back to Pāṇini's grammar of Sanskrit. However, the foundations of the *modern* dependency grammars were laid by Lucien Tesnière (Tesnière, 1959; Tesnière, 2015). On these grounds, a variety of dependency frameworks were proposed, such as Functional Generative Description (Sgall et al., 1986), Meaning Text Theory (Mel'čuk, 1988), Word Grammar (Hudson, 1984), or the

recent Universal Dependencies (Nivre et al., 2016). All these theories assume that syntactic structures of sentences can be modeled through dependency relations. They differ in aspects such as levels of representation, granularity of linguistic units, and properties of dependencies. We refer the reader to De Marneffe and Nivre (2019) for a comprehensive review of the key differences in the modern dependency grammar frameworks. In this dissertation, we focus on representations that reduce the structures to syntactic dependencies that hold between single words. Moreover, similarly to most of the dependency grammar frameworks, we make an assumption that the syntactic structure of sentence is a *tree* (Kübler et al., 2009).

2.1.1 Dependency Trees

Formal definition. Formally, we represent a sentence S as a sequence of tokens $[t_0, t_1, \ldots, t_n]$, where t_0 marks the auxiliary ROOT node and the subscripts from 1 to n correspond to the word order of the sentence.[1] Every t_i is a tuple $\langle f_1, f_2, \ldots \rangle$ of token-level atomic features, such as word forms, Part-of-Speech (POS) tags, or lemmas, i.e., canonical forms of the words. A *dependency tree* for a given sentence S is a pair $\langle S_V, A \rangle$, where S_V is a set of tokens of the sentence S, $A \subseteq S_V \times L \times S_V$ is a set of arcs, and L is a predefined set of possible labels for these arcs. A triple $\langle t_i, l, t_j \rangle$ represents a dependency relation $t_i \rightarrow t_j$ of the type l between the head token t_i and the dependent token t_j. We call the reflexive and transitive closure of the dependency relation the *dominance relation* and denote by $t_i \xrightarrow{*} t_k$ that t_i dominates t_k. In other words, $t_i \xrightarrow{*} t_k$ means that in the tree there is a directed path from t_i to t_k.

Three main constraints have to be fulfilled for a graph to be a well-formed dependency tree: (1) t_0 is the only token that does not have a head; (2) t_0 dominates every token in S, i.e., there is a directed path from t_0 to every token of the sentence; (3) for every token there is at most one head.

Graphical representation. Figure 2.1 presents a graphical representation of a dependency tree typical for the parsing literature. The tree consists of six nodes corresponding to the four words of the sentence "*He opened his eyes.*", the sentence-final punctuation mark, and an additional ROOT node introduced to uniquely identify the word that represents the root predicate of the sentence. Relations between the words are illustrated with directed, labeled arcs pointing from heads to their dependents. Labels of these arcs denote types of relations.

[1] We use the general term *word* to refer to tokens when it does not cause ambiguity.

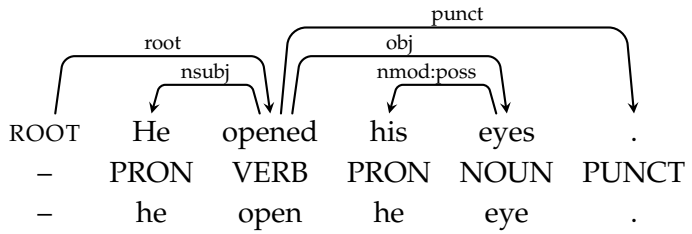


Figure 2.1: A projective dependency tree. The sentence (ID 2499) comes from the English LinES treebank (Ahrenberg, 2007).

The example tree consists of five dependencies: ⟨opened, *nsubj*, He⟩[2] identifies the word *He* as the nominal subject of this sentence; ⟨opened, *obj*, eyes⟩ informs that the word *eyes* fills the object role of the verb *opened*; ⟨eyes, *nmod:poss*, his⟩ means that the word *his* is a possessive nominal modifier of *eyes*; ⟨ROOT, *root*, opened⟩ marks the verb *opened* as the root of this sentence; ⟨opened, *punct*, .⟩ attaches the sentence-final dot to the root verb. This figure additionally includes information about the atomic features of the words. The first row contains POS tags, such as VERB for the word *opened* and NOUN for *eyes*. Below POS tags, we can see lemmas, e.g., *open* in the case of the word *opened*.

Projective vs. non-projective trees. The number of possible trees for a given sentence is exponential with respect to the size of the sentence. However, in naturally occurring structures, some constructions are more frequent than others. Therefore, parsing algorithms commonly put constraints on the structures that they can operate on, to reduce the complexity of the parsing problem (see Kuhlmann and Nivre (2006) and Havelka (2007) for a theoretical and empirical comparison of different tree constraints).

The most common constraint, i.e., projectivity, takes into consideration the relationship between the tree and the word order of the sentence. Let $\prec$ refer to the linear order of the words in a sentence. Formally, an arc is *projective* if the head dominates every intervening token between the head and the dependent:

$$t_i \rightarrow t_j \text{ is projective } \iff \begin{cases} \forall_{k,t_i \prec t_k \prec t_j} & t_i \xrightarrow{*} t_k \quad \textbf{if } t_i \prec t_j \\ \forall_{k,t_j \prec t_k \prec t_i} & t_i \xrightarrow{*} t_k \quad \textbf{if } t_j \prec t_i \end{cases} \tag{2.1}$$

We call a tree *projective* if all its arcs are projective and, consequently, *non-projective* if it contains at least one arc that is not projective. For example, Figure 2.2 shows a dependency tree for the sentence "*What do I need English for?*". The tree is non-projective because it contains the non-projective arc *What*→*for* (marked in green). The arc is not

[2]For convenience, we refer to tokens by their word forms.

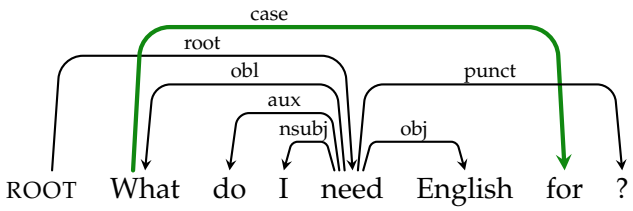


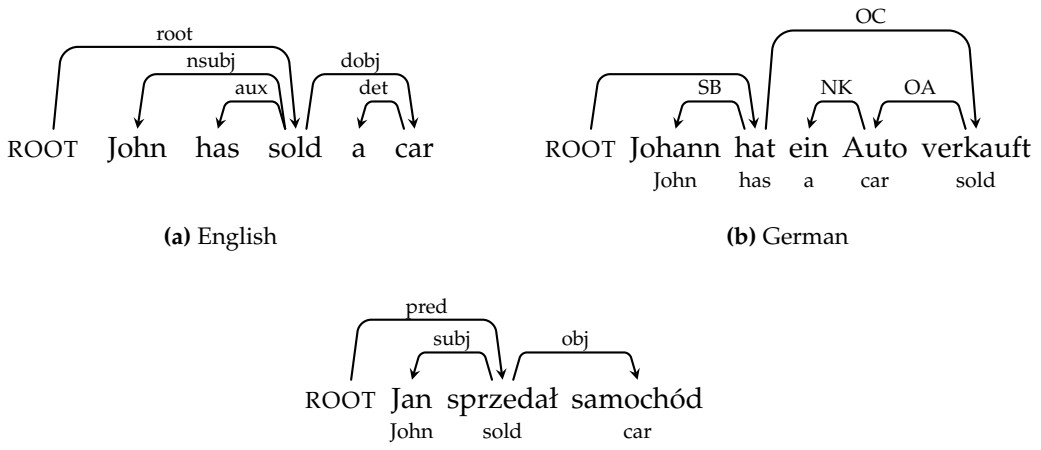
Figure 2.2: A non-projective dependency tree. The non-projective arc is marked in green. The sentence (ID 918) comes from the English LinES treebank.

projective since there is no path from the token *What* to such tokens as *need* or *English*.

Dependency parsers that restrict the search space of possible solutions to projective trees are very efficient (we give details on different parsing methods and their time complexity in Section 2.2). However, specific syntactic phenomena can be represented only with non-projective structures (Kuhlmann and Nivre, 2006). For example, the non-projectivity in the tree from Figure 2.2 is caused by a phenomenon called wh-movement. The phenomenon affects the placement of interrogative words, such as *What*, in English sentences. The frequency of non-projective arcs depends strongly on the annotation schema (see the next section), and language and its word order. Languages such as English or Spanish, which use relatively fixed word order, exhibit a limited number of non-projective arcs (see Havelka (2007) for statistics across twelve languages). On the other hand, in languages such as Czech or Turkish, which mark syntactic roles of words by declension rather than the position in the sentence, non-projective structures are much more common. For such languages, methods that perform non-projective parsing are preferable.

2.1.2 Dependency Treebanks

A *dependency treebank* is a corpus of sentences annotated with dependency trees. Such trees are either a result of manual annotation, as in the case of the Prague Dependency Treebank (Hajič et al., 2001), or of conversion from manually annotated trees for other parsing paradigms, as the constituency trees and English Penn Treebank (Marcus et al., 1993). In both cases, the trees follow a specific *annotation schema*, i.e., guidelines on how to apply a particular grammar theory to real-life sentences without running into ambiguity issues or inconsistent structures.



(a) English

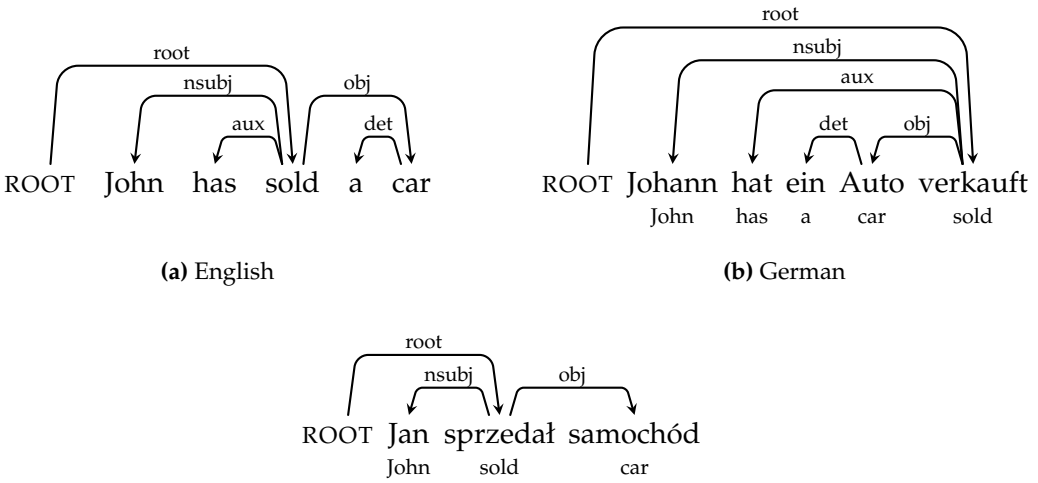
(b) German

(c) Polish

Figure 2.3: Parallel sentences annotated according to: (a) the English Penn Treebank schema converted to Stanford Dependencies (de Marneffe et al., 2014); (b) Seeker and Kuhn's (2012) conversion of the German TIGER schema; (c) the Polish Składnica schema (Woliński et al., 2011).

Language-specific treebanks. The above-mentioned Penn Treebank, together with the German TIGER (Brants et al., 2004) or the Polish Składnica (Świdziński and Woliński, 2010; Woliński et al., 2011), is an example of well-known treebanks and valuable language-specific resources that we use in this dissertation in Chapter 4. At the time of their release, all three of these treebanks facilitated research on statistical parsers for English, German, and Polish, respectively. However, since all of them were developed independently and with different languages in mind, their annotation schemata represent specific linguistic phenomena differently. For example, Figure 2.3 presents translations of the sentence "*John has sold a car*" and their analyses according to the annotation guidelines of the Penn Treebank converted to Stanford Dependencies (de Marneffe et al., 2014), Seeker and Kuhn's (2012) conversion of TIGER, and Składnica. We can notice incompatibilities among the three structures on the levels of labels and arcs. First, nominal subjects are annotated with three different labels: the English *nsubj*, the German *SB*, and the Polish *subj*. Second, the auxiliary verbs are treated differently, and in English, the word *sold* is the root of the sentence, while in German, the word *verkauft* (eng. sold) modifies the word *hat* (eng. has).

Parsing models that predict different language-specific representations are difficult to compare in a meaningful way. Moreover, such models cannot share knowledge about syntactic structures that are the same across languages.



(a) English

(b) German

(c) Polish

Figure 2.4: Parallel sentences from Figure 2.3 annotated according to the Universal Dependencies schema.

Universal Dependencies. Universal frameworks tackle the shortcomings of language-specific annotations. Such frameworks have been developed for many types of linguistic structures, such as morphological features (Sylak-Glassman et al., 2015) or semantic roles (Akbik et al., 2015). One of the first such initiatives were the universal POS tags introduced by Petrov et al. (2012). This universal tagset consisted of twelve Part-of-Speech categories and was initially applied to 22 different languages. It enabled comparing the tagging accuracy across languages. Moreover, it facilitated transferring dependency parsers between languages, i.e., training a model on sentences in one language and applying it to another.

The *Universal Dependencies (UD) project* (Nivre et al., 2016) is based on the universal POS tags and the Stanford dependencies (de Marneffe et al., 2014). It aims at creating a cross-linguistically consistent framework for dependency treebank annotation. So far, the project is the biggest initiative with the goal of creating resources annotated in a language-independent way. At the time of writing, it consists of 163 treebanks for 92 languages (ver. 2.6, Zeman et al. (2020)). Moreover, since the project is an on-going initiative, it constantly grows, with new or modified treebanks released every six months. In this dissertation, we use two different versions of the UD treebanks – ver. 2.0 (Nivre et al., 2017) and ver. 2.4 (Nivre et al., 2019) – depending on the available version at the time the experiments were performed.[3]

[3] All examples in this dissertation follow Universal Dependencies ver. 2.4 guidelines (Nivre et al., 2019).

To illustrate the main principles of the universal framework, Figure 2.4 presents the same sentences as in Figure 2.3 but annotated according to Universal Dependencies schema. We can notice three main differences between these two figures. First of all, since the project aims at developing unified representations, the labels of the arcs are normalized across languages. For example, nominal subjects are annotated with the relation *nsubj*, comparing to *SB* in TIGER or *subj* in Składnica. Secondly, UD dependency relations hold primarily between content words. We can observe that by comparing the two German trees in Figures 2.3b and 2.4b. According to the TIGER schema, the content words *Johann* (eng. John) and *verkauft* (eng. sold) are indirectly connected through the function word *hat* (eng. has). However, in the UD analysis, the word *Johann* is a direct modifier of *verkauft*. This annotation decision yields consistent syntactic structures with respect to the subject and object relations across the three languages, especially Polish, which does not need auxiliary verbs in this sentence. The third difference between the figures is that in the UD trees function words modify content words and not the other way around. See, for example, the two English trees and the relations ⟨has, *vc*, sold⟩ in Figure 2.3a and ⟨sold, *aux*, has⟩ in Figure 2.4a. This annotation choice is a consequence of the priority of the content words and distinguishes Universal Dependencies from other dependency frameworks (De Marneffe and Nivre, 2019).

2.2 Dependency Parsing

Dependency parsing is the task of finding a dependency tree for a given sentence. In this section, we start by putting the task in a broader context. We present a typical Natural Language Processing (NLP) architecture that employs a *dependency parser*, i.e., a system that performs dependency parsing, as one of its steps. We describe which annotations the parser needs as the input and discuss the relations between the parsing task and other tasks in the pipeline.

Next, we move to the methods for predicting dependency trees and present the two main parsing paradigms: transition-based and graph-based. We start from the traditional methods, i.e., algorithms that have been developed before the introduction of neural networks into the parsing field. Next, we address the more recent neural parsers and extensions that the deep learning brought into the parsing algorithms.

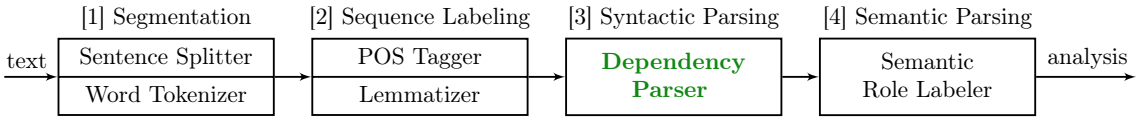


Figure 2.5: Schematic view of a typical NLP pipeline architecture. The architecture starts from raw text and produces a semantic analysis of the text (in this example, semantic roles). Dependency parsing, which is in the focus of this dissertation, is marked in green.

2.2.1 The Standard NLP Pipeline Architecture

The primary premise of pipeline architectures is that the tasks are solved in a step-wise manner: the later tasks use outputs of the previous ones and do not provide any feedback. Such an approach is presumably the most common in NLP applications due to its simplicity. However, it is sensitive to errors that propagate from the lower to the higher levels. Alternative solutions tackle this problem by performing two or more tasks jointly, e.g., POS tagging and parsing (Bohnet and Nivre, 2012), tokenization and tagging and parsing (Zhang et al., 2014), or sentence segmentation and parsing (Björkelund et al., 2016).

Figure 2.5 shows the architecture of a typical NLP pipeline, where the raw text is the input and semantic representations are the output. There are four steps in this particular pipeline. However, depending on the application, there might be further steps on top of the presented ones and some steps might be skipped. Figure 2.6 presents an output of the pipeline for a sentence "*John gave Mary some flowers.*". The four levels of the analysis are denoted by the numbers [1]–[4].

Segmentation.[4] The first step in Figure 2.5 includes word and sentence segmentation. This step defines basic linguistic units for further analysis. Word and sentence segmentation is often taken for granted in the parsing literature, and excluded from the experimentation setup. The reason is that for most datasets segmentation plays a minimal role, as it is delivered almost for free by means of whitespaces, sentence-final punctuation, and capital letters. For example, the first level in Figure 2.6 shows that segmentation, in this case, can be easily determined by whitespaces and punctuation. However, it is important to notice that there are applications in which segmentation is non-trivial. Such applications can roughly be grouped into two categories: (1) languages where tokenization is challenging, e.g., Chinese and Japanese, as well as languages such as Arabic and Hebrew, where many orthographic tokens are segmented into smaller syntactic words with

[4]This paragraph is based on the introduction to Section 2 from our publication (Björkelund et al., 2017).

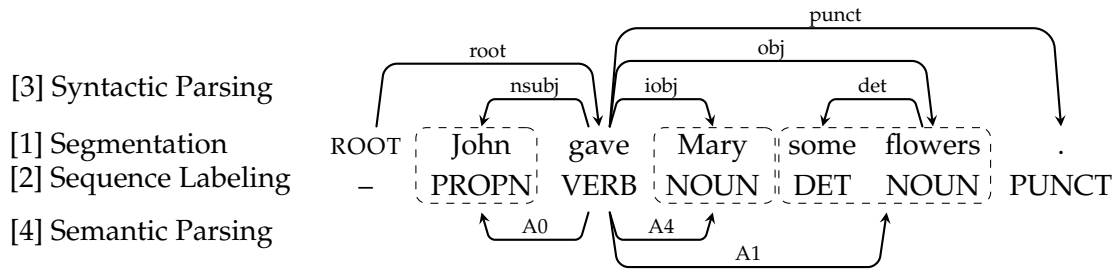


Figure 2.6: Result of the pipeline from Figure 2.5 for an example sentence "*John gave Mary some flowers.*" Borders of the arguments of semantic roles are marked with dashed boxes.

transformations; (2) applications where the detection of sentence boundaries is difficult, such as classical texts.

In this dissertation, we follow the typical setting and assume correct sentence segmentation and tokenization in all our experiments. Therefore, the methods that we use have to be preceded by tools for word and sentence segmentation to parse raw text.

Sequence labeling. The next three steps presented in Figure 2.5 produce different levels of linguistic structure, from POS tags up to semantics. Sequence labeling, which is the first of these steps, is the simplest form of parsing (cf., the broad definition of parsing in Jurafsky and Martin (2014, Ch. 3)). It assigns a word category from a predefined set, such as POS tags or morphological features, to each word in a sequence.

In the second level in example Figure 2.6, the POS tags are assigned to the given tokens. Even in this simple example, we observe that the quality of sequence labelers directly influences the performance of the following, more complex tasks. For example, the POS tagging decision of labeling *some* as DET (rather than PRON) is interrelated with the structural decision of making it a dependent of *flowers* in the dependency syntax.

Semantic parsing. Dependency trees can be used by a variety of downstream tasks, such as named entity recognition, relation extraction, translation, or natural language generation (Kübler et al., 2009, p.1). Here, we take semantic parsing as an example. It is the task of predicting deeper, semantic analyses for a given sentence. The task has recently received increased attention as syntactic analyses are not necessarily expressive enough to be exploited by downstream applications, a fact that was also empirically corroborated in the shared task on Extrinsic Parser Evaluation (Oepen et al., 2017).

Semantic role labeling, i.e., the task of identifying and labeling predicate-argument structures, is an example of semantic parsing tasks presented in Figure 2.5. Such struc-

Algorithm 2.1 Greedy transition-based parsing with a statistical model.

Input: Sentence S
Input: Weight vector w
1: **function** PARSETB(S, w):
2: $c \leftarrow$ INITCONFIGURATION(S)
3: **while** $\neg$TERMINAL(c) **do**
4: $\hat{t} \leftarrow \underset{t \in \text{VALID}(c)}{\arg\max}\ \text{SCORE}(c, t, w)$
5: $c \leftarrow \hat{t}(c)$
6: **return** BUILDTREE(c)

tures are typically limited to verbal and nominal predicates (two most popular annotation schemes were developed in projects PropBank (Palmer et al., 2005) and FrameNet (Baker et al., 1998)) and share many interrelations with dependency structures. For example, A0 arguments (agents) tend to be realized as *nsubj* dependents, and A1 arguments (themes) as *obj* (see Figure 2.6). Therefore, a semantic role labeling system can directly exploit information from dependency trees predicted on the lower pipeline level.

2.2.2 Transition-based Parsing

We now turn to the first class of dependency parsers – *transition-based models*. As mentioned earlier, we first focus on the traditional, non-neural parsers, which use linear classifiers and discrete, manually designed feature models. In Section 2.2.4, we describe how these two aspects change when neural networks are used.

Data-driven transition-based parsers were independently developed by Yamada and Matsumoto (2003) and Nivre (2003). They were motivated by shift-reduce parsing, which gradually builds the final tree in a step-wise manner. Algorithm 2.1 presents a general view of transition-based parsing. We first give a brief description of the algorithm and provide missing details in the subsequent sections. The function PARSETB takes as an input a sentence S and a trained weight vector w.[5] It then prepares an initial configuration c (line 2). This configuration is built only from the tokens of the input sentence S and, in the beginning, contains no arcs. Next, a sequence of transitions is applied to the configuration c (line 5). Each of such transitions has the highest score among all the valid ones (line 4). Finally, when the terminal state is reached (line 3), the resulting tree is built from the terminal configuration (line 6).

[5]For now, we assume that the weight vector (or matrix) is given and focus on how to use it to parse a sentence. Section 2.3 covers issues related to training statistical models.

Before we go into the details of Algorithm 2.1, it is vital to notice that the presented parsing procedure is *greedy*. The parser always selects and applies the transition that is locally the best (line 4). Such an approach is efficient but can suffer from error propagation, i.e., errors at early steps cannot be recovered from and influence the subsequent decisions. Therefore, the traditional transition-based parsers achieve robust performance by employing methods to tackle the error-propagation problem, such as *beam search* (Zhang and Clark, 2008). The approach involves keeping a list (beam) of several (k) best scoring sequences of transitions. At every parsing step, each of the sequences is expanded by every possible transition, scored, and only the k best scoring sequences are kept. That way the method considers more than one derivation and can recover from errors. However, its complexity depends also on the beam size k and is $O(nk)$, where n is the length of the sentence.

2.2.2.1 Transition Systems

Transition-based parsers use *transition systems* to formalize how to derive the dependency tree in a finite number of steps, going from an initial configuration to the terminal one. The two first transition systems introduced by Nivre (2003, 2004) – ARCSTANDARD and ARCEAGER – can produce only projective trees. Both of them are sound and complete, i.e., they can only derive projective trees and, for any projective tree, there is a sequence of transitions to derive it (Nivre, 2008). Moreover, both are very efficient and can build the final tree in linear time with respect to the given sentence's size. The main difference between the two systems is that the ARCSTANDARD introduces an arc only if the dependent node has already collected all its children, while the ARCEAGER system builds the tree in an eager way, adding an arc in the earliest possible step.

Following work defined other transition systems, like the arc-hybrid system (Kuhlmann et al., 2011), easy-first parser (Goldberg and Elhadad, 2010), or the LR-spine system (Sartorio et al., 2013). In this dissertation, we use a system called SWAPSTANDARD (Nivre, 2009), an extended version of the ARCSTANDARD system that can deal with non-projective trees. The theoretical time complexity of the system is quadratic with respect to the size of the sentence. However, Nivre (2009) empirically showed that in practice, it runs in linear time. Below we give details on this particular system.

Configurations. Configurations are triples $\langle \sigma, \beta, A \rangle$, where σ is a stack of partially processed tokens, β is a buffer of input tokens, and A is a set of so-far created arcs. The initial configuration is represented by the triple $\langle [t_0], [t_1, t_2, ..., t_n], \emptyset \rangle$, where t_0 represents

Transition					Preconditions
LEFTARC$_l$	$\langle \sigma \mid s_1 \mid s_0, \quad \beta, A \rangle$	$\rightarrow$	$\langle \sigma \mid s_0, \quad \beta, A \cup \{\langle s_0, l, s_1 \rangle\} \rangle$		$s_1 \neq$ ROOT
RIGHTARC$_l$	$\langle \sigma \mid s_1 \mid s_0, \quad \beta, A \rangle$	$\rightarrow$	$\langle \sigma \mid s_1, \quad \beta, A \cup \{\langle s_1, l, s_0 \rangle\} \rangle$		
SHIFT	$\langle \sigma, \quad b_0 \mid \beta, A \rangle$	$\rightarrow$	$\langle \sigma \mid b_0, \quad \beta, A \rangle$		
SWAP	$\langle \sigma \mid s_1 \mid s_0, \quad \beta, A \rangle$	$\rightarrow$	$\langle \sigma \mid s_0, \; s_1 \mid \beta, A \rangle$		$s_1 \neq$ ROOT $\wedge$ $s_1 \prec s_0$

Table 2.1: Transitions of the SWAPSTANDARD system (Nivre, 2009).

an artificial ROOT node and t_i are tokens in their original sentence order. During parsing, a sequence of transitions is applied to such a configuration until the final stage $\langle [t_0], [], A \rangle$ is reached, i.e., until the buffer is empty and the stack contains only the ROOT token. At that point, the set A contains arcs which encode the full parse tree for the given sentence.

Transitions. Transitions are partial functions which map one configuration into another. The SWAPSTANDARD system comprises of four transitions listed in Table 2.1. The transitions come with additional preconditions (right column in the figure), which define for which configurations they are valid (such preconditions directly translate into the VALID function in Algorithm 2.1). We use an operator $\mid$ to separate the head from the tail of the stack or the front from the back of the buffer (the operator is right- and left-associative, respectively). Moreover, s_i denotes elements of the stack and b_i elements of the buffer.

The first three transitions make up the ARCSTANDARD system. SHIFT moves tokens from the front of the buffer onto the stack. Both LEFTARC$_l$ and RIGHTARC$_l$ introduce arcs between the two top-most items of the stack s_0 and s_1. LEFTARC$_l$ comes with an additional precondition that the second top-most item of the stack is not ROOT. Otherwise, the system would allow adding an arc with ROOT as a dependent, which would violate the first tree constraint that we saw in Section 2.1.1, i.e., restriction that ROOT does not have a head. Both transitions LEFTARC$_l$ and RIGHTARC$_l$ are parametrized for the label l of the arc that they introduce.

The fourth transition SWAP extends the ARCSTANDARD system and allows it to produce non-projective trees. The transition re-orders the tokens by pushing the second top-most item of the stack back to the buffer. The transition has two preconditions. The first precondition assures that the token pushed back to the buffer is not ROOT. According to the second tree constraint ROOT dominates all tokens in the sentence. Therefore, arcs originating from ROOT are always projective, and there is no need to put ROOT in the buffer. The second precondition makes sure that the two top-most items of the stack are

Algorithm 2.2 The lazy SWAP oracle from Nivre et al. (2009), where:
$A(x)$ denotes a set of all modifiers of x in A,
$\prec_{PROJ}$ denotes projective order,
MPC(x) returns an identifier of the maximal projective component of x.

Input: Configuration $c = \langle \sigma | s_1 | s_0, b_0 | \beta, A \rangle$
Input: Correct set of arcs A_c

```
1: function LAZYORACLE(c, A_c):
2:     if ⟨s_0, l, s_1⟩ ∈ A_c and A_c(s_1) = A(s_1) then
3:         return LEFTARC_l
4:     if ⟨s_1, l, s_0⟩ ∈ A_c and A_c(s_0) = A(s_0) then
5:         return RIGHTARC_l
6:     if s_0 ≺_PROJ s_1 and MPC(s_0) ≠ MPC(b_0) then
7:         return SWAP
8:     return SHIFT
```

in linear order. That way, a token that was swapped already once will not be swapped again, and the algorithm will avoid infinite swaps.

2.2.2.2 Oracle Parsing

Training transition-based parsers on manually annotated trees requires *an oracle*, i.e., an algorithm which for a given tree and a configuration returns the next correct transition. In this dissertation, we use the lazy SWAP oracle proposed by Nivre et al. (2009), i.e., an improved version of the original SWAP oracle for the SWAPSTANDARD system (Nivre, 2009). It is a static oracle, which means that it defines a constant (canonical) order of transitions in which they are selected (cf., the concept of a dynamic oracle (Goldberg and Nivre, 2012, 2013)). That way, it tackles a problem called spurious ambiguity – although in the SWAPSTANDARD system it is possible to derive the same tree with different sequences of transitions, the oracle always returns one transition in a deterministic way.

The lazy SWAP oracle. Algorithm 2.2 presents the oracle function LAZYORACLE, which for a given configuration $c = \langle \sigma | s_1 | s_0, b_0 | \beta, A \rangle$ and a set of correct arcs A_c returns the next correct transition. We can notice that the canonical order of transitions in this oracle is LEFTARC$_l$, RIGHTARC$_l$, SWAP, and SHIFT. The first two of them come with straightforward conditions that assure that the introduced arc is correct, and that the token which will be removed from the configuration after the transition (s_1 for LEFTARC$_l$ and s_0 for RIGHTARC$_l$) has already collected all its modifiers (the modifiers in A_c and A are the same).

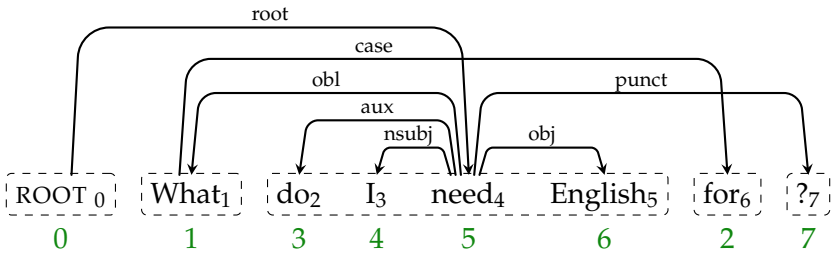


Figure 2.7: Dependency tree from Figure 2.2 annotated with projective order (green) and maximal projective components (dashed boxes). Subscripts denote the positions of tokens in the sentence.

Initial configuration			⟨ [0],	[1 2 3 4 5 6 7],	∅ ⟩
1	SHIFT	→	⟨ [0 1],	[2 3 4 5 6 7],	∅ ⟩
2	SHIFT	→	⟨ [0 1 2],	[3 4 5 6 7],	∅ ⟩
3	SHIFT	→	⟨ [0 1 2 3],	[4 5 6 7],	∅ ⟩
4	SHIFT	→	⟨ [0 1 2 3 4],	[5 6 7],	∅ ⟩
5	LEFTARC$_{nsubj}$	→	⟨ [0 1 2 4],	[5 6 7],	{⟨4, *nsubj*, 3⟩} ⟩
6	LEFTARC$_{aux}$	→	⟨ [0 1 4],	[5 6 7],	{⟨2, *aux*, 4⟩, ⟨4, *nsubj*, 3⟩} ⟩
7	SHIFT	→	⟨ [0 1 4 5],	[6 7],	{⟨2, *aux*, 4⟩, ⟨4, *nsubj*, 3⟩ } ⟩
8	RIGHTARC$_{obj}$	→	⟨ [0 1 4],	[6 7],	{⟨4, *obj*, 5⟩, ⟨2, *aux*, 4⟩, . . .} ⟩
9	SHIFT	→	⟨ [0 1 4 6],	[7],	{⟨4, *obj*, 5⟩, ⟨2, *aux*, 4⟩, . . .} ⟩
10	SWAP	→	⟨ [0 1 6],	[4 7],	{⟨4, *obj*, 5⟩, ⟨2, *aux*, 4⟩, . . .} ⟩
11	RIGHTARC$_{case}$	→	⟨ [0 1],	[4 7],	{⟨1, *case*, 6⟩, ⟨4, *obj*, 5⟩, . . .} ⟩
12	SHIFT	→	⟨ [0 1 4],	[7],	{⟨1, *case*, 6⟩, ⟨4, *obj*, 5⟩, . . .} ⟩
13	LEFTARC$_{obl}$	→	⟨ [0 4],	[7],	{⟨4, *obl*, 1⟩, ⟨1, *case*, 6⟩, . . .} ⟩
14	SHIFT	→	⟨ [0 4 7],	[],	{⟨4, *obl*, 1⟩, ⟨1, *case*, 6⟩, . . .} ⟩
15	RIGHTARC$_{punct}$	→	⟨ [0 4],	[],	{⟨4, *punct*, 7⟩, ⟨4, *obl*, 1⟩, . . .} ⟩
16	RIGHTARC$_{root}$	→	⟨ [0],	[],	{⟨0, *root*, 4⟩, ⟨4, *punct*, 7⟩, . . .} ⟩
Final configuration					

Figure 2.8: Oracle parsing result for the tree in Figure 2.7 and the lazy SWAP oracle. For better readability, we present tokens as their positions in the sentence [ROOT$_0$, What$_1$, do$_2$, I$_3$, need$_4$, English$_5$, for$_6$, ?$_7$] and show only the two items most recently added to the set of arcs. Two configurations referred to in the main text are underlined.

The goal of the third transition SWAP is to re-order input tokens so that non-projective arcs could be introduced. The lazy SWAP oracle defines two conditions for this transition.

The first condition ($s_0 \prec_{\text{PROJ}} s_1$) allows performing SWAP only on tokens, which are not in projective order. Nivre (2009) defines $\prec_{\text{PROJ}}$ as an in-order traversal of the correct tree while respecting the surface order between nodes and their direct dependents. For example, Figure 2.7 presents the non-projective tree from Figure 2.2 with tokens annotated with positions in the projective order (marked in green). Re-ordering these tokens according to the given order would yield a sentence "*What for do I need English?*" and the corresponding tree would be projective.

An oracle that uses only the first condition performs SWAP as early as possible and makes a lot of unnecessary transitions. Nivre et al. (2009) proposed to postpone performing SWAP until bigger structures are built (this is why we refer to this oracle as "lazy"). The oracle uses an auxiliary concept of a maximal projective component (MPC). MPCs can be obtained by performing the oracle parsing without the SWAP transition. Running such an oracle on a non-projective tree yields a configuration where the set of arcs contains multiple projective trees instead of a single, fully connected one. Figure 2.7 illustrates such MPCs with boxes around tokens. The second condition for the SWAP transition allows SWAP only if s_0 and b_0 do not belong to the same MPC. As a result, unnecessary swapping is limited.

Example oracle sequence. Figure 2.8 presents a full oracle sequence of transitions for the example tree from Figure 2.7. To improve readability, we present tokens only as their positions in the sentence and show only the two items most recently added to the set of arcs. The sequence starts from an initial configuration and ends after sixteen transitions with the final configuration (the buffer is empty and the stack contains only ROOT). We bring attention to the two most interesting configurations in this sequence (underlined with dashed lines). After the sixth transition, the two top-most items of the stack are What_1 and need_4. Although the tree contains the arc $\langle \text{need}_4, \textit{obl}, \text{What}_1 \rangle$ $\text{LEFTARC}_{\textit{obl}}$ is not the correct transition for this configuration. The reason is that the second constraint for LEFTARC_l is not fulfilled, i.e., the token What_1 is still missing for_6 as a dependent. SWAP is also not correct for this configuration because the tokens need_4 and English_5 (s_0 and b_0, respectively) are a part of the same MPC. Finally, SHIFT is selected, which is the fallback option.

The second interesting configuration is three rows below, after the ninth transition. The two top-most items are need_4 and for_6. Since these tokens are not in projective order (positions 5 and 2, respectively) and they are not part of the same MPC, the transition SWAP is correct for this configuration.

$\phi_i(c,t) = 1$	iff	*dog* is the word form of b_0 in c
$\phi_j(c,t) = 1$	iff	NOUN is the POS tag of s_1 in c
or		
$\phi_k(c,t) = 1$	iff	*cat* is the word form of s_0 and VERB is the POS tag of b_0 in c

Figure 2.9: Example indicator functions for a given configuration c and a transition t.

2.2.2.3 Transition Scoring and Feature Extraction

The last part of Algorithm 2.1 that still needs explanation is the function SCORE and the concept of scoring transitions.

Scoring transitions. In Algorithm 2.1 (line 4), we saw that the parser scores all possible transitions and selects the best one to apply. The scoring function SCORE takes as an input the current configuration c and a trained prior to parsing weight vector $w \in \mathbb{R}^m$.

The traditional, non-neural parsers used in this dissertation use linear models, i.e., they calculate scores as the scalar product between the weight vector w and the vector representation of the given configuration $\phi(c,t)$:

$$\text{SCORE}(c,t,w) \leftarrow w \cdot \phi(c,t) \tag{2.2}$$

Feature function. The *feature function* ϕ maps configurations into high-dimensional, real-valued vectors: $\phi(c,t) \in \mathbb{R}^m$. Designing such functions requires substantial manual effort. On the one hand, such functions have to encode as much information about the configuration as possible to allow the parser to make well-informed decisions. On the other hand, this information should be as generic as possible so that the parser can generalize well and make decisions for configurations that it has not seen before.

Feature functions are usually implemented as high-dimensional binary vectors. Each dimension in such a vector corresponds to an individual indicator function (see examples of indicator functions in Figure 2.9). Such individual indicators are constructed in two steps: designing feature templates and filling them with possible values.

The first manual step requires designing *feature templates*. Such templates consist of two types of information: configuration positions and token attributes. Configuration positions inform which items of the stack, buffer, and so-far predicted arcs will take part in the feature extraction (see, for example, b_0 in ϕ_i or s_1 in ϕ_j in Figure 2.9). Usually, much attention is paid to the top-most stack items, as they are the tokens that LEFTARC$_l$ and RIGHTARC$_l$ introduce arcs for, and the front of the buffer, since this is the next token to

process and possibly SHIFT. However, features drawn from the parsing history, i.e., so-far predicted arcs, are also very important because they give additional structural context for making the right decisions. Moreover, since the parser builds trees gradually and extends the set of predicted arcs in a step-wise manner, the amount of structural information that the feature function has access to grows at every step.

Token attributes are the second type of information included in feature templates. Such attributes usually encode the linguistic properties of the tokens, such as POS tag, morphological features, or lemmas (see, for example, POS tag in ϕ_j in Figure 2.9). However, token attributes can also depend only on surface forms and contain, for example, information if the word is capitalized or if it ends with a particular suffix.

The second step in constructing individual feature indicators is to populate them with possible *feature values* (e.g., *dog* in ϕ_i or NOUN in ϕ_j in Figure 2.9). This step is automatic. More specifically, during training, the parser performs oracle parsing on training sentences and extracts all possible unique feature values for the designed feature templates. It then maps every such feature value into one indicator function.

Feature model. *The feature model* is a collection of feature templates designed for a particular parser. Table 2.2 presents an example feature model from Zhang and Nivre (2011) (for simplicity, we present only the baseline feature model from this work). The model was designed for a parser using the arc-eager system (Nivre, 2003). In this system, arcs are introduced between the top item of the stack and the first item of the buffer. This is why the model puts more emphasis on b_0 than on s_1.

The model uses only two types of token attributes – word forms (left column), POS tags (middle), or concatenation of both of them (right column). We use the notation POS(x,y) for a concatenation of POS tags (or word forms) of tokens at the positions x and y. Feature templates are divided into three groups: unigram features, which look at only one configuration position at the time, and bigram and trigram features, which introduce combinations of two or three positions. Such combination features are very important, especially when linear classifiers are used, because they introduce interrelations between features (see more about (non-)linear classifiers and feature combinations in Section 2.2.4.1).

Among the configuration positions in Table 2.2, we notice the top-most item of the stack (denoted s_0) and the three first items of the buffer (b_0, b_1, b_2). Trigram features also use structural positions, such as left- and rightmost dependents of s_0 and b_0 (denoted $._L$ and $._R$, respectively) and the head of s_0 (denoted $._H$). However, these features use only POS tags and no lexical attributes. Otherwise, the model would introduce many rare

Unigram features		
form(s_0)	POS(s_0)	form(s_0) ∘ POS(s_0)
form(b_0)	POS(b_0)	form(b_0) ∘ POS(b_0)
form(b_1)	POS(b_1)	form(b_1) ∘ POS(b_1)
form(b_2)	POS(b_2)	form(b_2) ∘ POS(b_2)
Bigram features		
form(s_0, b_0)	POS(s_0, b_0)	form(s_0, b_0) ∘ POS(s_0, b_0)
	POS(b_0, b_1)	form(s_0, b_0) ∘ POS(s_0)
		form(s_0, b_0) ∘ POS(b_0)
		form(s_0) ∘ POS(s_0, b_0)
		form(b_0) ∘ POS(s_0, b_0)
Trigram features		
	POS(b_0, b_1, b_2)	
	POS(s_0, b_0, b_1)	
	POS(s_{0H}, s_0, b_0)	
	POS(s_0, s_{0L}, b_0)	
	POS(s_0, s_{0R}, b_0)	
	POS(s_0, b_0, b_{0L})	

Table 2.2: Example feature model from Zhang and Nivre (2011). s_i and b_i denote i-th elements of the stack and buffer, respectively; $\cdot_L$ and $\cdot_R$ denote left- and rightmost dependent of an item and $\cdot_H$ its head; POS(x,y) is a conjunction (also denoted by ∘) of POS tags (or word forms) of tokens at the positions x and y.

feature indicators, and the resulting feature vectors would be very high-dimensional and sparse.

2.2.3 Graph-based Parsing

We now switch our focus from the transition-based parsing paradigm to the *graph-based* models (Eisner, 1996; McDonald et al., 2005a). Algorithm 2.3 presents a general view of graph-based parsing. We can immediately notice that the algorithm is much shorter than the one for the transition-based parsing (cf., Algorithm 2.1) and can be expressed as a single line. It is because graph-based models do not build trees gradually. Instead, they define a space of all possible dependency trees for a given sentence S (denoted $\mathcal{Y}(S)$) and search for the tree in $\mathcal{Y}(S)$, that has the highest score. Below, we first provide details on how the graph-based parsers score trees. Next, we describe how they search through

Algorithm 2.3 Graph-based parsing with a statistical model;
$\mathcal{Y}(S)$ denotes the space of all possible dependency trees for S

Input: Sentence S
Input: Weight vector w
1: **function** PARSEGB(S, w):
2: $\hat{y} \leftarrow \arg\max_{y \in \mathcal{Y}(S)} \text{SCORE}(y, w)$
3: **return** $\hat{y}$

$\mathcal{Y}(S)$ in an effective way.

2.2.3.1 Tree Scoring and Feature Extraction

Graph-based parsers use scoring functions to represent how likely it is that a particular tree is a correct analysis for the given sentence. To calculate the scores efficiently, the scoring functions apply *factorization*: (1) first, the given tree is factored into smaller subgraphs, (2) then each of such subgraphs is scored independently, and (3) the partial scores are summed up to give the final score.

First-order vs. higher-order models. The simplest, *arc-factored models* define the scores of trees directly by the scores of their arcs:

$$\text{SCORE}(y, w) = \sum_{\langle h,l,d \rangle \in y} \text{SCORE}(\langle h, l, d \rangle, w) \tag{2.3}$$

Such arc-factored models score arcs independently from each other, therefore, they discard much useful information. For example, Figure 2.10 compares dependency trees for two very similar sentences "*The pudding fell under the table with a huge hole.*" and "*The pudding fell under the table through a huge hole.*". The two trees differ by one arc (marked in green). When an arc-factored model is used, the decision about these arcs must be made without the knowledge about other arcs in the tree. Specifically, scoring the arcs *fell→hole* and *table→hole* has to be performed without information which of the *case* dependents the token *hole* has – *with* or *through*. In such short sentences as in Figure 2.10 this type of information can be also captured by expressive feature templates. However, when the involved tokens are far away from each other, only looking at their surface context might not be enough.

Both of the above-described examples emphasize the importance of relations between two arcs: *fell→hole→through* and *table→hole→with*. Models that consider factors built

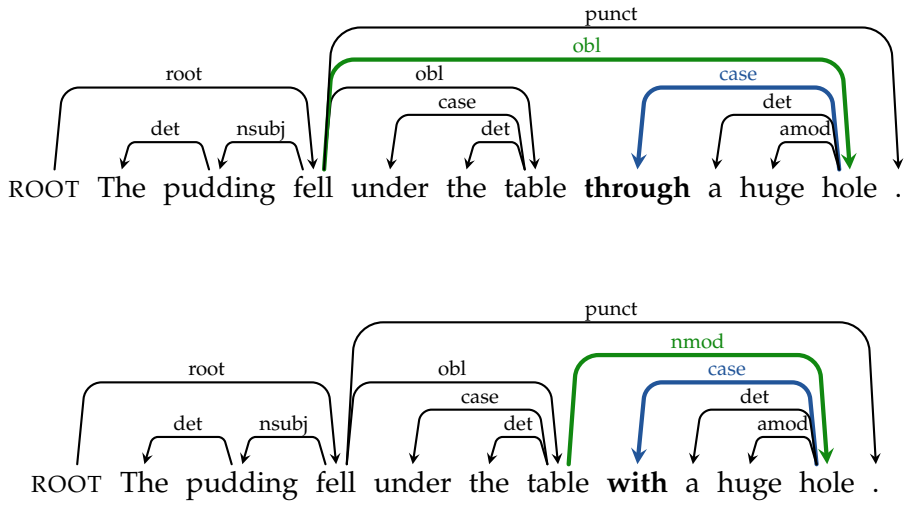
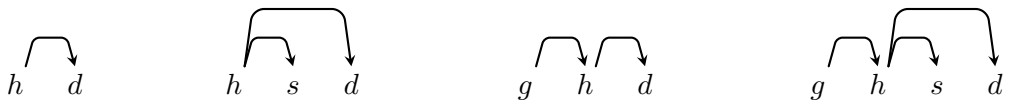


Figure 2.10: An example of second-order factors. Differences between the two trees are marked in green.



(a) First order **(b)** Second order (sibling) **(c)** Second order (grandchild) **(d)** Third order (grand-sibling)

Figure 2.11: First- and higher-order factors as presented by Zhang and Zhao (2015).

from two arcs are called *second-order*. The arcs in the second-order factors can be in the grandparent relation, as in the examples above, or in the sibling relation, such as ⟨hole, *case*, with⟩ and ⟨hole, *det*, a⟩. Figure 2.11 compares the two second-order factors with a third-order factor, which takes into consideration both grandparents and siblings, and a single arc, which for consistency is commonly called a first-order factor. The higher the order of the factor, the more information it encodes, and the better the performance of the parser. However, models that depend on complex factorization come with higher time complexity. We address this issue in Section 2.2.3.2.

Scoring factors. Scoring single factors is performed in an analogous way to scoring transitions in the transition-based models (see Section 2.2.2.3). The function SCORE first represents the given factor through a feature function ϕ as a high-dimensional real-valued vector. It then uses a (usually) linear classifier and computes the score as a scalar product between the feature vector and the weight vector. Therefore, for the arc-factored

model from Equation (2.3) scores are calculated as:

$$\textsc{Score}(\langle h, l, d\rangle, w) \leftarrow w \cdot \phi(\langle h, l, d\rangle) \tag{2.4}$$

Feature extraction. The main difference between scoring transitions and factors is the scope of ϕ. In the transition-based models, the function has access to the whole given configuration. Especially, it can draw complex structural features from the parsing history and so-far predicted partial trees. In the case of graph-based models, the scope of ϕ is restricted to the given factor. Although it can build features from all sentence tokens, the only *structural relations* that it can look at are the ones in the given factor.

We illustrate the distinction between transition- and graph-based feature functions with an example feature model presented in Table 2.3. The model comes from McDonald et al. (2005a) and was designed for an arc-factored graph-based parser. As attributes, it takes only word forms and POS tags. Feature templates are divided into three groups depending on how many tokens they take into consideration. The first group consists of unigram features, which look only at heads or dependents. Bigram features include both of the tokens and build more or less all possible combinations of their word forms and POS tags. Finally, the third group adds additional tokens from the sentence. However,

	Unigram features	
form(h)	POS(h)	form(h) ∘ POS(h)
form(d)	POS(d)	form(d) ∘ POS(d)
	Bigram features	
form(h, d)	POS(h, d)	form(h, d) ∘ POS(h, d)
		form(d) ∘ POS(h, d)
		form(h, d) ∘ POS(d)
		form(h, d) ∘ POS(h)
		form(h) ∘ POS(h, d)
	Other features	
	POS(h, b, d)	
	POS(h, d, $h_{\pm1}$, $d_{\pm1}$)	

Table 2.3: Exemplary feature model from McDonald et al. (2005a) for Eisner's decoder; h denotes head, d dependent, b any token between the head and dependent, ±1 tokens after/before head or dependent. All features are combined with the direction of the edge (left, right) and the distance between head and dependent.

these additional tokens are defined by the surface relation to the head and dependent and not a structural one. For example, the first template POS(h, b, d) looks at tokens b between h and d, and the second one, POS(h, d, $h_{\pm 1}$, $d_{\pm 1}$), at tokens at positions one to the left or right from h and d (denoted ± 1). When we compare this feature model with the model from Table 2.2, we can immediately notice that fewer tokens are taken into consideration, and relations between them are limited.

2.2.3.2 Graph-based Decoders

The second part of the graph-based parsing procedure presented in Algorithm 2.3 that needs explanation is the $\arg\max$ function. We said that the parser searches through $\mathcal{Y}(S)$, i.e., the space of all possible dependency trees for the given sentence S, and returns the one with the highest score. Since the size of $\mathcal{Y}(S)$ is exponential in the length of the input sentence, simply scoring all possible trees and searching through them would not be tractable. Instead, parsers use *graph-based decoders*, i.e., search algorithms which find the highest scoring tree in an efficient way. We now briefly describe the two graph-based decoders used in this dissertation: the Chu-Liu-Edmonds algorithm and Eisner's algorithm. For a more detailed description we refer the reader to Kübler et al. (2009).

The Chu-Liu-Edmonds algorithm. McDonald et al. (2005b) cast the dependency parsing problem as a search for the Maximum Spanning Tree (MST) in a fully-connected directed graph. They used the Chu-Liu-Edmonds (CLE) algorithm (Chu and Liu, 1965; Edmonds, 1967), which for a given graph and a scoring function finds the spanning tree with the highest sum of arc scores. With Tarjan's (1977) implementation, the algorithm can run in quadratic time with respect to the number of graph nodes.

The Chu-Liu-Edmonds algorithm presented in Algorithm 2.4 is greedy and recursive. Since its pseudocode is quite complicated, we illustrate it through an example adapted from Kübler et al. (2009, p.48). Moreover, we omit labels of arcs for simplicity.

Figure 2.12 presents stages of the algorithm applied to a sentence "*The pudding fell*" (we shorten the sentence from Figure 2.10 to only three words for better readability). The algorithm starts from an almost fully-connected directed graph in which nodes are tokens of the sentence, and edge weights are the results of the SCORE function (Figure 2.12a). The only missing edges are the ones that would enter the ROOT node and are therefore unnecessary. To apply the function CLE to a sentence it is necessary to first convert it into such a graph.

In the first greedy step, the algorithm selects the highest-scoring incoming arc for

Algorithm 2.4 The Chu-Liu-Edmonds algorithm for finding the maximum spanning tree in a given graph;
See Algorithm 2.5 for pseudocode of CONTRACT and Algorithm 2.6 for RESOLVECYCLE;
Function FINDCYCLE(G) returns a set of nodes that forms a cycle in G.

Input: Graph $G = \langle V, A \rangle$, where V is a set of nodes and A is a set of arcs
Input: Weight vector w

```
1:  function CLE(G = ⟨V, A⟩, w):
2:      A' ← {⟨h, d⟩ | d ∈ V, h = arg max_{h∈V} SCORE(⟨h, d⟩, w)}   ▷ Select best heads greedily
3:      G' ← ⟨V, A'⟩
4:      C ← FINDCYCLE(G')
5:      if C is empty then                                          ▷ MST was found
6:          return G'
7:      else
8:          G_C ← CONTRACT(G, C, w)                                 ▷ Contract the cycle
9:          T ← CLE(G_C, w)                                         ▷ Call CLE recursively
10:         return RESOLVECYCLE(T,C)                                ▷ Resolve the cycle
```

each node (thick arcs on Figure 2.12b and line 2 in Algorithm 2.4). If, after this step, there is no cycle among the selected edges, they form a proper, maximum-scoring tree, and the algorithm has found its result (line 5 in Algorithm 2.4). However, if there is a cycle, as between nodes *The* and *pudding* in Figure 2.12b, the algorithm moves to the next step. The tokens that constitute the cycle are contracted into one new node (v_c in Figure 2.12c and line 8 in Algorithm 2.4). The scores for arcs entering and leaving the new node are recalculated so that they represent the state before the contraction (see Algorithm 2.5). In the next step, the algorithm recursively applies the CLE procedure and finds the maximum spanning tree for the new smaller graph (thick arrows in Figure 2.12d and line 9 in Algorithm 2.4). Finally, all contracted nodes are resolved so that the result is a spanning thee for the original graph (Algorithm 2.6). For example, in Figure 2.12e, the node v_c unfolds back to the two nodes *The* and *pudding*. The final tree contains the arc *fell*→*pudding* because the smaller spanning tree contained the arc *fell*→v_c, and the path *fell*→*pudding*→*The* scores higher than *fell*→*The*→*pudding*.

Algorithm 2.5 Part of the Chu-Liu-Edmonds algorithm (see Algorithm 2.4) responsible for contracting cycles into single nodes;
Function HEAD(x) returns the head of the node x.

Input: Graph $G = \langle V, A \rangle$, where V is a set of nodes and A is a set of arcs
Input: Set of nodes C that forms a cycle
Input: Weight vector w

function CONTRACT($G = \langle V, A \rangle$, C, w):
  $G_C = G - C$ ▷ Subgraph of G excluding nodes in C
  Create v_c to represent C

  for $d \in V - C : \exists_{h \in C} \langle h, d \rangle \in A$ **do** ▷ Compute scores for arcs leaving v_c
    Add $\langle v_c, d \rangle$ to G_C
    Set SCORE($\langle v_c, d \rangle, w$) $\leftarrow \max_{h' \in C}$ SCORE($\langle h', d \rangle, w$)

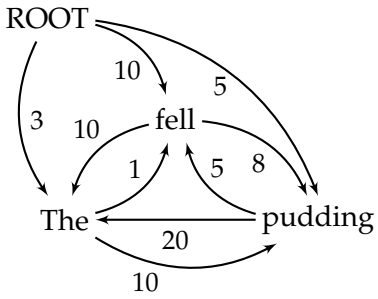
  for $h \in V - C : \exists_{d \in C} \langle h, d \rangle \in A$ **do** ▷ Compute scores for arcs entering v_c
    Add $\langle h, v_c \rangle$ to G_C
    $s \leftarrow \max_{d' \in C} \big($SCORE($\langle h, d' \rangle, w$) $-$ SCORE($\langle$HEAD(d'), $d' \rangle, w$)$\big)$
    Set SCORE($\langle h, v_c \rangle, w$) $\leftarrow s + \sum_{t \in C}$ SCORE($\langle$HEAD(t), $t \rangle, w$)

  return G_C

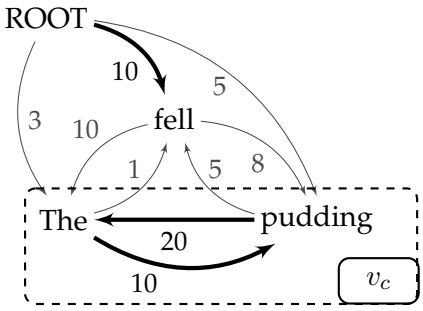
Algorithm 2.6 Part of the Chu-Liu-Edmonds algorithm (see Algorithm 2.4) responsible for resolving contracted cycles.

Input: Tree $T = \langle V, A \rangle$, where V is a set of nodes and A is a set of arcs
Input: Set of nodes C that forms a cycle

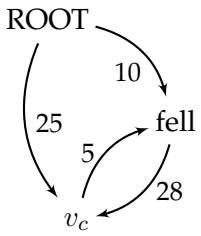
function RESOLVECYCLE(T, C):
  Find a node d in C s. t. $\langle h', d \rangle \in T$ and $\langle h'', d \rangle \in C$
  return $T \cup C - \{\langle h'', d \rangle\}$



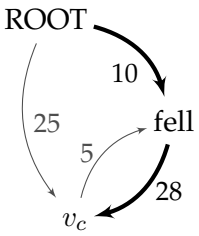
(a) Fully connected directed graph with arc scores.



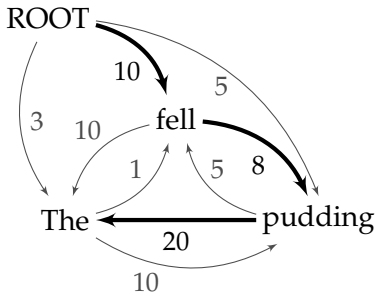
(b) Result of the greedy selection of highest-scoring heads (thick arrows). Dashed lines mark a cycle.



(c) The smaller graph with the cycle replaced with a new node v_c.



(d) Result of the recursive run of the CLE algorithm on the smaller graph. Thick arrows mark MST.



(e) The final result of the algorithm. v_c gets resolved back into the previous nodes taking into consideration the smaller MST.

Figure 2.12: Illustration of the Chu-Liu-Edmonds algorithm on an example sentence "*The pudding fell*". An example adapted from Kübler et al. (2009, pp. 48).

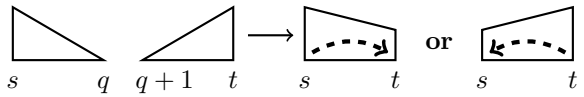
Eisner's algorithm. Eisner (1996) developed an efficient bottom-up algorithm for projective dependency parsing. Later, McDonald et al. (2005a) presented how to effectively train a statistical dependency parser that uses Eisner's decoder. The algorithm is based on two main observations. First, a projective dependency tree spanning a sentence can be broken down into two subtrees spanning two adjacent parts of the sentence and an arc between their roots. This observation allows for approaching dependency parsing with dynamic programming. More specifically, the algorithm is an adaptation of the Cocke-Kasami-Younger (CKY) algorithm (Cocke, 1969; Kasami, 1966; Younger, 1967) for context-free parsing. A direct application of CKY to dependency parsing would require a complexity of $O(n^5)$. But due to the second observation behind the algorithm, which is that the tokens can collect its left dependents independently of the right ones, the algorithm can run in cubic time.

To explain how the Eisner's decoder works, we follow McDonald et al. (2005a) and use geometric figures to illustrate sequences of consecutive tokens of the input sentence. Figure 2.13 shows four types of structures that Eisner's algorithm uses to represent a subtree that spans tokens between positions s and t. Each span contains only one token that does not have a head, which is on the periphery of the structure (longer vertical edge of the polygon). In closed structures (left), only the head token is active and can still collect new dependents. In open structures (right), both tokens (s and t) are active.

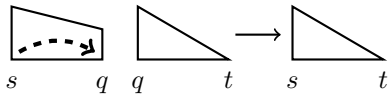
Algorithm 2.7 presents pseudocode for this procedure. The algorithm starts by creating four matrices that will keep scores of four types of structures (O stands for open, C for closed, l and r for left and right, respectively). The parsing algorithm starts by representing every token in the input sentence as two closed structures (left and right) and assigning them zero scores (line 4). It then considers longer and longer spans of tokens and finds the highest-scoring structures (one of every type) for all of them (lines 5 to 11). New structures are created by following the rules presented in Figure 2.14: (1) combining two adjacent closed structures with heads on the opposite sides creates an open structure and introduces an arc between their roots; (2) combining closed and open structures with heads on the same side creates a closed structure. A score for a new structure is the sum of scores of the two smaller component structures and, in the case of open structures, the score of the introduced arc. After the algorithm fills the whole dynamic programming table, the score of the left-closed structure, which spans the whole sentence, is the score of the best tree for the given sentence (line 12). The final step of Eisner's algorithm, after running the function from Algorithm 2.7, is to backtrack through the dynamic table and find the tree that results in the best found score (we provide an example of such backtracking in Appendix B.1).

(a) right-closed **(b)** left-closed **(c)** right-open **(d)** left-open

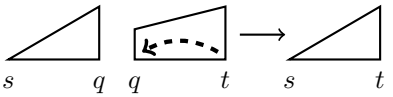
Figure 2.13: Types of structures built by Eisner's algorithm. The structures represent spans of tokens between positions s and t. Heads of the structures are marked by the longer edge.



(a) Combining two closed structures gives an open one and introduces an arc.



(b) Combining left-open and left-closed structures gives a left-closed structure.



(c) Combining right-closed and right-open structures gives a right-closed structure.

Figure 2.14: Rules for combining structures in Eisner's algorithm.

Algorithm 2.7 Eisner's algorithm for finding the maximum spanning projective tree for a given sentence.

Input: Sentence S with n tokens
Input: Weight vector w

function EISNER(S,w):
  Instantiate $O_r[n][n]$, $O_l[n][n]$, $C_r[n][n]$, $C_l[n][n] \in \mathbb{R}$

  for each token index s in S **do**
    Initialize $O_r[s][s]$,$O_l[s][s]$,$C_r[s][s]$,$C_l[s][s] \leftarrow 0.0$

  for m from $1 \ldots n$ **do**
    for s from $0 \ldots n - m - 1$ **do**
      $t \leftarrow s + m$
      $O_r[s][t] \leftarrow \max_{s \leq q < t} C_l[s][q] + C_r[q+1][t] + \text{SCORE}(\langle t, s \rangle, w)$
      $O_l[s][t] \leftarrow \max_{s \leq q < t} C_l[s][q] + C_r[q+1][t] + \text{SCORE}(\langle s, t \rangle, w)$
      $C_r[s][t] \leftarrow \max_{s \leq q < t} C_r[s][q] + O_r[q][t]$
      $C_l[s][t] \leftarrow \max_{s < q \leq t} O_l[s][q] + C_l[q][t]$

  return $C_l[0][n]$

Chu-Liu-Edmonds vs. Eisner's algorithm. The two described graph-based decoders have their strengths and weaknesses. The Chu-Liu-Edmonds algorithm performs an exact search through the space of all possible trees. Thus, parsers that use this decoder perform non-projective parsing inherently. In contrast, the basic version of Eisner's algorithm is restricted to projective trees and has to be extended to perform non-projective parsing. For example, McDonald and Pereira (2006) presented an approximation algorithm that allows for recovering non-projective dependencies. The algorithm first finds the highest-scoring projective tree and then rearranges its edges until the introduced changes stop improving the score of the tree.

The second difference between the two decoders is their runtime complexity and access to higher-order features. Parsers using the Chu-Liu-Edmonds algorithm can be effective and run in quadratic time with respect to the length of the sentence. However, their feature functions are limited to single arcs, because higher-order non-projective MST parsing is NP-hard (McDonald and Pereira, 2006; McDonald and Satta, 2007). Eisner's algorithm, as introduced by Eisner (1996), can also be applied only to arc-factored models. However, as shown by Zhang and McDonald (2012), the algorithm can be extended to incorporate higher-order factors while keeping its polynomial complexity. McDonald and Pereira (2006) were the first to demonstrate such an extension and increased the scope of features to pairs of adjacent edges while staying in the $O(n^3)$ complexity class. In the following years, higher-order extensions were proposed. The algorithm of Carreras (2007) uses second-order grandchildren factorization, and the variant proposed by Koo and Collins (2010) can incorporate features over three arcs. Both of the methods come with a higher runtime complexity of $O(n^4)$.

The runtime complexity of parsers incorporating higher-order features is a consequence of performing an exact search through the space of all possible trees. Filtering some of the possible solutions from this search space allows for performing higher-order parsing more efficiently. Such filtering can be performed, for example, through a multi-pass run of the first-, second-, and third-order models (Rush and Petrov, 2012) or cube pruning (Zhang and McDonald, 2012). Koo et al. (2010) presented a different approach and proposed an approximate parsing method in which second-order (or third-order as in (Martins et al., 2013)) features are modeled with head automata. The method uses a dual decomposition algorithm to ensure that the CLE decoder, which enforces the tree constraint, and the component automata agree.

2.2.4 Neural Dependency Parsers

In the previous sections, we looked at the classical transition- and graph-based dependency parsers invented before the introduction of deep learning techniques into the parsing field. Now, we turn our attention to the neural parsers. We focus on the most crucial inventions from the perspective of the neural parsing architectures used in this dissertation: non-linear classifiers, distributed word representations, and representations encoding sentential context. We refer the reader to Goldberg (2017) for a broader description of neural networks and their application to a variety of Natural Language Processing tasks.

2.2.4.1 Neural Networks and Non-linear Functions

The traditional parsers that we have described so far used linear classifiers. Specifically, the scoring functions from Equations (2.2) and (2.4) depended linearly on the feature values. Such linear models have their limitations – they cannot introduce interrelations between features or fit data that is not linearly separable.

Overcoming limitations of linear classifiers. One way of dealing with the limitations of linear models is to map the data into very high-dimensional space that is linearly separable and train a linear classifier on such mapped representations. For example, one of the first implementations of transition-based parsers by Yamada and Matsumoto (2003) used Support Vector Machines (SVMs), i.e., a statistical classifier which performs such mapping through kernel methods (Boser et al., 1992). More specifically, they used polynomial kernels and showed that models that use combinations of two features (second-degree polynomials) give better performance than these that use only single ones. However, as later stated by Attardi (2006), parsers that use this classifier are computationally expensive. Since the prediction step in SVMs depends on the size of the training set, they are impractical when parsers are trained on big treebanks.

The parsers that we saw in the previous sections deal with limitations of the linear models differently. They introduce feature combinations, i.e., feature templates that involve more than one feature indicator, such as the bigram and trigram features in Tables 2.2 and 2.3. That way, the non-linear relations between features are added to the representations, and the classifier can learn which of them are important for the final decision. However, such combination features have to be either picked manually, based either on the designer's intuition or on the time-absorbing feature engineering, or induced through complex architectures (Titov and Henderson, 2007).

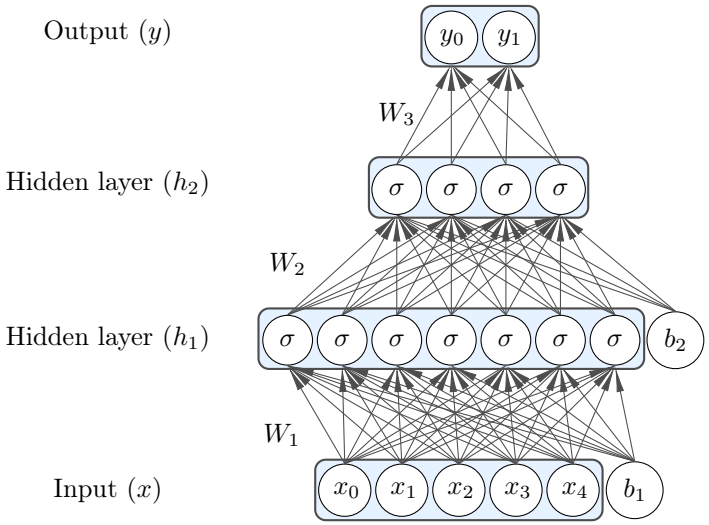


Figure 2.15: A feed-forward neural network with two hidden layers.

Multi-layer Perceptron (MLP). Feed-forward neural networks, also known as *multi-layer perceptrons*, are powerful statistical classifiers that rely on *non-linear functions*. Attardi et al. (2009) were ones of the first who employed an MLP in a parsing architecture. Their goal at the time was mostly to improve the parsing speed of an SVM-based parser. Later, Chen and Manning (2014) popularized the models in the field by showing that they allow learning the important combinations of features automatically.

Figure 2.15 presents a common graphical representation of a neural network. This type of network is commonly referred to as *feed-forward* because the connections between nodes (or neurons) do not form a cycle, and the information flows through them in one direction. Each circle in the figure represents a neuron. Information flows in and out of such neurons through the arrows. Additionally, every arrow carries a weight (single weights are not shown in the figure, they are grouped into weight matrices W_1, W_2, and W_3 for better readability).

The example network in Figure 2.15 consists of four layers: an input layer (a vector x), two hidden layers (h_1 and h_2), and an output layer (a vector y). The input layer is a high-dimensional real-valued vector (in the figure, it has five dimensions and a bias unit b_1). Each neuron in the hidden layer calculates its output by multiplying the input values by their weights, summing them, and applying a non-linear function (marked by σ). The second hidden layer performs the same steps with the input from the first hidden layer. Finally, the last layer of the network is its output (in the figure, the output has two dimensions).

Since each layer implements a vector-matrix multiplication, the computation performed by the two-layer perceptron in Figure 2.15 can be written as

$$\text{MLP}_2(x) = y, \tag{2.5}$$

where

$$\begin{aligned} h_1 &= \sigma_1(W_1 \cdot x + b_1) \\ h_2 &= \sigma_2(W_2 \cdot h_1 + b_2) \\ y &= W_3 \cdot h_2. \end{aligned} \tag{2.6}$$

W_1, W_2, and W_3 are weight matrices; σ_1 and σ_2 are non-linear functions; b_1 and b_2 represent bias units. There is a wide variety of non-linear functions that can be used in neural models. As explained by Goldberg (2017, pp.45-46), the decision which function to use in a particular task is usually an empirical question. For example, the above-mentioned architecture of Chen and Manning (2014) employed a cube function to introduce relations between three elements of the vectors. The functions that we use and refer to in this dissertation are presented in Table 2.4.

Incorporation of a multi-layer perceptron into transition-based dependency parsers is rather straightforward and requires replacing the linear scoring function from Equation (2.2) with MLP:

$$\text{SCORE}(c, t, w) \leftarrow \text{MLP}(\phi(c, t), w) \tag{2.7}$$

Analogously, for graph-based parsers the function from Equation (2.4) has to be replaced

Cube	$\sigma(x) = x^3$
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$
Hyperbolic Tangent (tanh)	$\sigma(x) = \frac{e^{2x}-1}{e^{2x}+1}$
Rectified Linear Unit (ReLU)	$\sigma(x) = \begin{cases} 0 \text{ for } x < 0 \\ x \text{ otherwise} \end{cases}$

Table 2.4: Examples of non-linear functions used in neural networks.

with MLP to score arcs:

$$\text{SCORE}(\langle h, l, d\rangle, w) \leftarrow \text{MLP}(\phi(\langle h, l, d\rangle), w) \tag{2.8}$$

However, it is important to notice that now w stands for all the weight matrices W and biases b that are involved in the calculations in Equation (2.6). Moreover, we omitted the subscript in MLP because the decision of how many hidden layers to use in a particular parser is an empirical one (see page 43).

2.2.4.2 Distributed Word Representations

The first dependency parsers that used neural networks, such as the architecture of Attardi et al. (2009), did not necessarily come with higher accuracy than their non-neural precedents. However, they gave grounds for subsequent deep learning extensions, such as *distributed word representations*, also known as dense word embeddings. Models that use such representations map words into real-valued vectors and train the vectors together with other parameters. Such trained vectors can capture syntactic and semantic similarities between words, which, as a result, improves the generalization power of the models (Collobert et al., 2011). Below we describe first transition- and graph-based parsers that used such distributed word representations. Next, we provide additional background on methods to pre-train such representations on large samples of unannotated text.

Transition-based parser. Chen and Manning (2014) were the first to incorporate dense representations into a transition-based dependency parser. Their parser used a simple greedy search, the ARCSTANDARD system, and a multi-layer perceptron. The authors replaced the traditional sparse indicator features, which were still used by Attardi et al. (2009), with a small number of dense vectors. Their parser was not only faster than a parser that used SVMs but also achieved state-of-the-art results at that time.

Figure 2.16 illustrates Chen and Manning's (2014) parser scoring transitions for an example configuration $\langle$ [t_0, t_1, t_3], [t_5, t_6, t_7], $\{\langle t_3, t_2\rangle, \langle t_3, t_4\rangle\}\rangle$ (we omit the labels of arcs for simplicity). At the bottom of the architecture, we present the tokens $[t_1, \ldots t_7]$ of the sentence, the root token t_0, and the configuration positions of these tokens. For example, t_1 is the second top-most item of the stack (s_1) and t_4 is annotated with s_{0R} since it is the right dependent of t_3, the top-most item of the stack.

The first level of the architecture comprises of dense word representations x_i (from now on we illustrate vectors with rectangles). In Chen and Manning's (2014) approach

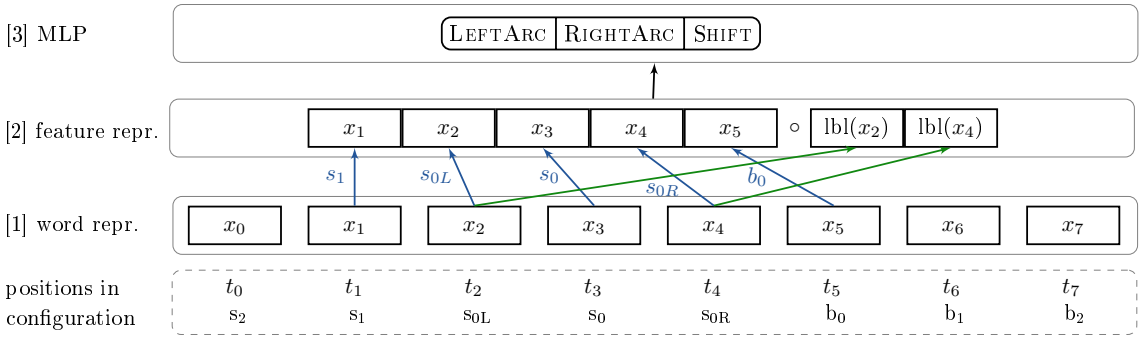


Figure 2.16: Schematic illustration of Chen and Manning's (2014) architecture scoring transitions for an example configuration $\langle [t_0, t_1, t_3], [t_5, t_6, t_7], \{\langle t_3, t_2\rangle, \langle t_3, t_4\rangle\}\rangle$.

each x_i is built from concatenating embeddings of the word form and its POS tag:

$$x_i = e(\text{form}(t_i)) \circ e(\text{POS}(t_i)) \tag{2.9}$$

In the next step, the parser selects a few core positions from the given configuration as feature vectors (blue arrows in the figure). To improve the readability of Figure 2.16, we present only five such positions: the two top-most items of the stack (s_1, s_0), front item of the buffer (b_0), and left- and rightmost children of the top item of the stack (s_{0L}, s_{0R}). The actual implementation of Chen and Manning (2014) uses 18 feature vectors: the three top-most items of the stack and buffer, the first and second left- and rightmost children of s_0 and s_1, the leftmost children of s_{0L} and s_{1L}, and the rightmost children of s_{0R} and s_{1R}. The feature representation (level [2] on the figure) is a concatenation of all feature vectors and dependency label embeddings (function lbl) of the features coming from so-far predicted arcs (s_{0L} and s_{0R} on the figure and green arrows). Such feature representation is an input to an MLP consisting of one hidden layer with a cube activation function and a softmax layer. The softmax function converts the outputs of the network into a probability distribution over all the possible decisions:

$$\text{softmax}(y)_i = \frac{e^{y_i}}{\sum_j e^{y_j}} \tag{2.10}$$

Finally, MLP assigns scores to all possible transitions.

Graph-based parser. Pei et al. (2015) adapted Chen and Manning's (2014) architecture to a graph-based setting. They proposed two models using either first-order or second-order factorization, and Eisner's decoder. Scoring factors was performed with a neural

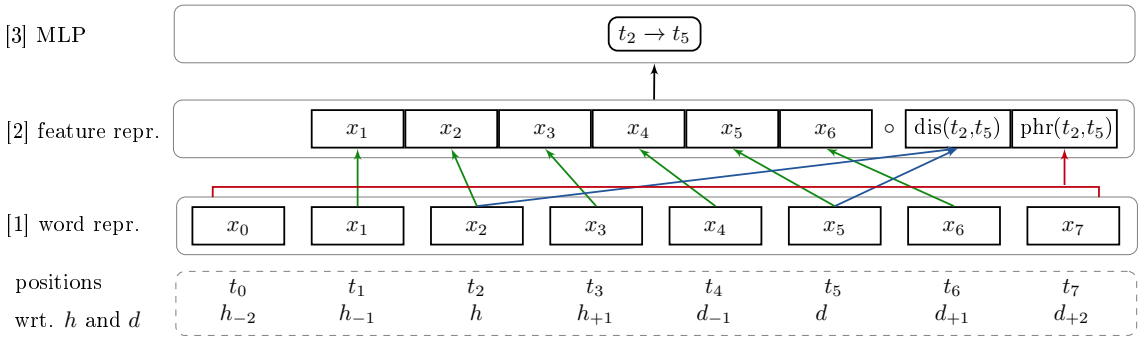


Figure 2.17: Schematic illustration of Pei et al.'s (2015) first-order architecture scoring an arc $\langle t_2, , t_5 \rangle$.

architecture similar to the one that we saw for the transition-based parser.

Figure 2.17 gives an example of Pei et al.'s (2015) first-order architecture scoring an arc $\langle t_2, t_5 \rangle$ for a sentence $[t_1, \ldots, t_7]$. At the bottom of the figure, we provide the tokens of the sentence and information at which positions they are with respect to the considered arc. For example, the tokens t_2 and t_5 are annotated with h (for head) and d (for dependent), t_4 received d_{-1} because it is one position to the left from the dependent, and t_3 is one position to the right from the head and is annotated h_{+1}.

The first level of the architecture, i.e., word representations, is the same as in the transition-based parser. Representations x_i are built from concatenating embeddings of words and their POS tags. Next, ten feature vectors are selected (the figure shows only six of them, see green arrows): vectors of the head (h), the dependent (d), and words at positions ± 1 and ± 2 from h and d. Additionally, the feature representation incorporates information about the distance between the head and dependent (dis(h, d), blue arrows) and a phrase embedding (phr(h, d), red arrow). The goal of the last component is to add a global context to the representation. It is a concatenation of three vectors: an average of embeddings of words before h (x_0, x_1), between h and d (x_3, x_4), and after d (x_6, x_7).

The last step of the architecture is analogous to the architecture of Chen and Manning (2014). The feature vectors are concatenated and passed to a multi-layer perceptron. MLP consists of one hidden layer with proposed by the authors *tanh-cube* activation function, which is a composition of a cube and a hyperbolic tangent functions. Finally MLP assigns a score to the given arc.

Pre-trained word embeddings. In the case of the parsers described above, incorporation of dense representations resulted in higher accuracy and improved parsing time. However, distributed word embeddings come with yet another benefit – they can be in-

duced from large amounts of unannotated text prior to training. Already Chen and Manning (2014) showed that such pre-trained word embeddings improve the performance of dependency parsers. For pre-training, they used the `word2vec` method, which is based on the skip-gram with a negative sampling model (Mikolov et al., 2013). The method builds representations of words based on their context in a sentence. Later, many modifications have been introduced to the `word2vec` model to make the final embeddings more suited for syntactic tasks. For example, Ling et al. (2015) added into the model the information about the relative positioning of context words, and Kanerva et al. (2017) built the embeddings only from syntactic contexts, without words themselves. Both methods resulted in embeddings that improved the performance of dependency parsers more than basic `word2vec` representations.

2.2.4.3 Sentential Context

To correctly score a transition or an arc, a parser must take into consideration directly involved words, such as the two top-most items of the stack in transition-based parsing, and words next to them. Such neighboring words can be crucial for making correct parsing decisions. For example, in Figure 2.10, we saw that the context word *through* is crucial for scoring the arc ⟨fell, *obl*, hole⟩. The architectures that we saw in the previous section encode such sentential context through rather big feature sets – 18 feature vectors in the case of Chen and Manning (2014), and ten feature vectors and three phrase vectors in (Pei et al., 2015). Such feature sets have to be manually selected, which does not guarantee that the most important words for making a particular decision are included.

RNNs. Recurrent Neural Network (RNN) is a neural architecture that directly incorporates the context of the whole sentence into the word representation. Intuitively, it can be seen as the MLP that we saw in Figure 2.15 adapted to processing sequential data.

Figure 2.18 (right) presents the Simple Recurrent Neural Network (SRNN), specifically Elman network (Elman, 1990). The network accepts an input sequence of vectors $[x_0, \ldots, x_t]$ and produces a sequence of outputs $[y_0, \ldots, y_t]$:

$$RNN(x_{0:n}) = y_{0:n} \tag{2.11}$$

A single output y_t is a function of two elements: the input x_t and an additional, internal state h_{t-1}. This state can be seen as a summary of all the information seen so far. That way, decisions of SRNNs are influenced not only by the current input but also by all the previous ones. In the case of the Elman network, the vector h_t is the result of the hidden

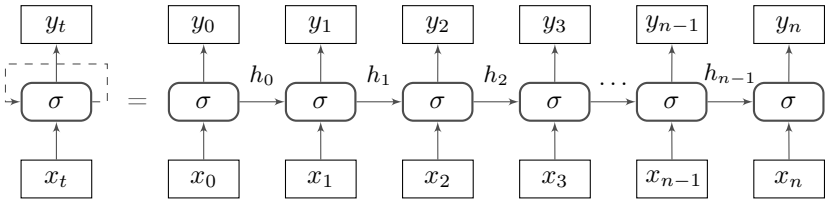


Figure 2.18: Graphical representation of a simple recurrent neural network; left shows a recursive representations; right shows an unrolled version. Based on Bjerva (2017, p.33).

layer from the previous step:

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \tag{2.12}$$
$$y_t = \sigma_y(W_y h_t + b_y),$$

where W_h and W_y are weight matrices for the current input and output, respectively; U_h stores weights for the state vector h_{i-1}; σ_h and σ_y are non-linear functions; b_h and b_y represent bias units. After the output y_i is calculated, h_i is passed to the next step. This recursive step is the reason why the architectures are commonly represented in a folded way (see Figure 2.18, left).

LSTMs. The internal state of an SRNN is a result of multiple applications of a non-linear function. During training, the loss information that is backpropagated through all these non-linear functions can vanish before reaching the first steps, which is known as the vanishing gradient problem (Pascanu et al., 2013). Thus, although theoretically, SRNNs can capture long-distance dependencies, training them to do so is difficult (refer to Section 2.3 for more information about training neural models).

The Long Short-Term Memory (LSTM) networks (Hochreiter and Schmidhuber, 1997) are a variant of RNNs designed to deal with this problem. The architectures, similarly to SRNNs, keep an internal state which is updated after every input. However, LSTMs use three additional gates – forget, input, and output – to control which information should be retained within the internal state and which should be forgotten.

Specifically, LSTM is a function mapping a sequence of inputs $[x_0, \ldots, x_n]$ to a se-

quence of outputs $[y_0, \ldots, y_n]$. Each of the outputs y_t takes the following form:

$$
\begin{aligned}
f_t &= \sigma(W_f x_t + U_f y_{t-1} + b_f) \\
i_t &= \sigma(W_i x_t + U_i y_{t-1} + b_i) \\
o_t &= \sigma(W_o x_t + U_o y_{t-1} + b_o) \\
\\
\tilde{c_t} &= \sigma_c(W_c x_t + U_c y_{t-1} + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c_t} \\
y_t &= o_t \odot \sigma(c_t),
\end{aligned}
\tag{2.13}
$$

where: matrices W and U store weights; vectors b represent bias units; σ is a non-linearity; $\odot$ denotes element-wise product. Moreover, f_t is the output of the forget gate, i_t the input gate, and o_t the output gate. These three vectors depend on the current input x_t and the output of the previous state y_{t-1}. Outputs of the forget and input gates are used to control the information in a memory cell c_t. More specifically, the cell is a combination of two terms. The first one uses the forget gate to control which information from the previous steps to keep ($f_t \odot c_{t-1}$). The second one decides which of the new information should be stored in memory. It does it by multiplying the input gate's output by an activation vector of the current input ($i_t \odot \tilde{c_t}$). Finally, the output y_t depends on the current memory cell and the output gate o_t. The gate additionally controls which of the memory information should be used to calculate the output.

LSTM-based parsers. Dyer et al. (2015) were the first to use LSTMs within a parsing architecture. They presented a transition-based dependency parser, which outperformed the architecture of Chen and Manning (2014) on two treebanks. The parser used three stack-LSTMs, i.e., LSTMs extended with push and pop operations, to represent a stack, a buffer, and a list of so-far performed transitions. Within this architecture, not only tokens but also partial trees were represented as dense representations. Dyer et al. (2015) achieved that by recursively applying a composition function to heads and determiners. In the following work, Kiperwasser and Goldberg (2016a) showed how to perform such composition through tree-LSTMs, i.e., LSTMs adapted to model trees.

The line of research initialized by Dyer et al. (2015) and Kiperwasser and Goldberg (2016a) focused on designing new parsing architectures that make better use of the structural context present in the parsing state. A different approach was independently presented by Kiperwasser and Goldberg (2016b) and Cross and Huang (2016), who proposed

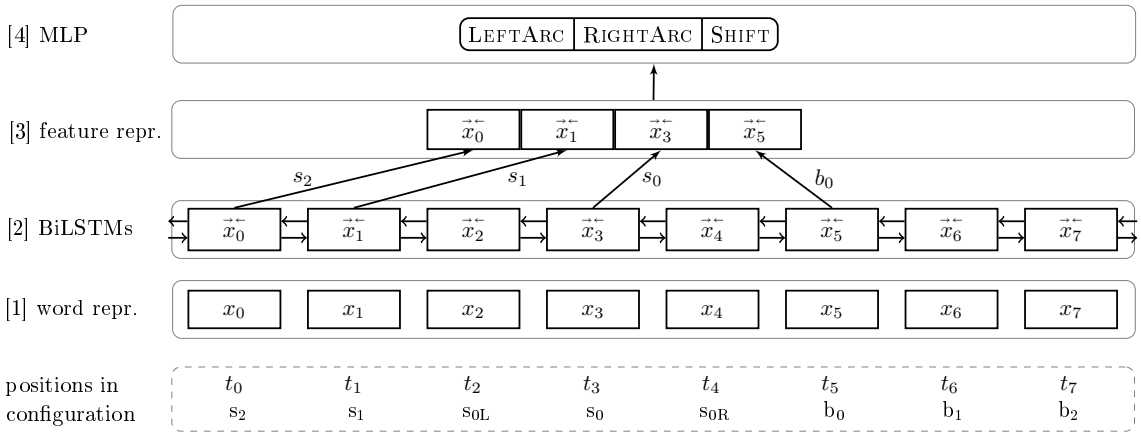
to use LSTMs as feature extractors. Here we focus on the work of Kiperwasser and Goldberg (2016b), who presented the neural transition-based and graph-based architectures employed in this dissertation. These architectures can be generally seen as the models of Chen and Manning (2014) and Pei et al. (2015) with an additional layer of LSTMs. The layer incorporates sentential context into the word representations before they are passed to the final MLP.

K&G architectures. Figure 2.19 illustrates the architectures of Kiperwasser and Goldberg (2016b) (we refer to these architectures as **K&G**) in an analogous way to Figures 2.16 and 2.17. In both transition- and graph-based approaches, input tokens are represented in the same way as in architectures of Chen and Manning (2014) and Pei et al. (2015) – see level [1] in the figures. For a given sentence $[t_1, \dots t_n]$ each word representation x_i is built from concatenating the embeddings of the word form and its POS tag. These representations x_i encode words in isolation and do not contain information about their context. For that reason, they are passed to the BiLSTM feature extractors (level [2] in Figure 2.19) and represented by a BiLSTM representation $\vec{x_i}$. BiLSTMs, i.e., Bi-Directional Long Short-Term Memory networks (Graves and Schmidhuber, 2005), are extended LSTMs that model the left- and right-context of each word. They are usually implemented as two separate LSTMs: forward LSTM, which reads the input sentence from the beginning, and backward LSTM, which processes words in a reversed order:

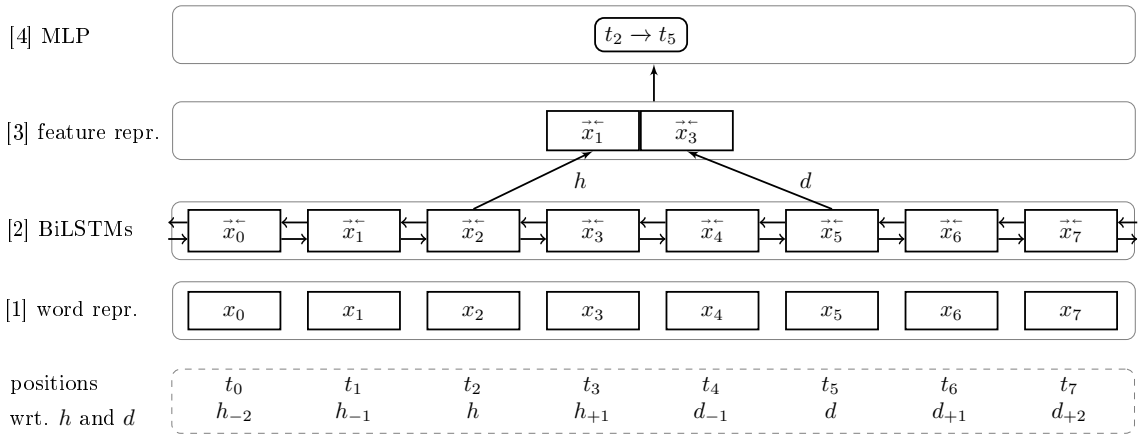
$$\vec{x_i} = \text{BiLSTM}(x_{0:n}, i) = \text{LSTM}(x_{0:i}) \circ \text{LSTM}(x_{n:i}) \tag{2.14}$$

The feature extraction step (layer [3]) looks very similar to the architectures of Chen and Manning (2014) and Pei et al. (2015). The parser selects a few core items from the sentence as features. Next, it concatenates their BiLSTM vectors and passes to MLP. The perceptron assigns scores to all possible transitions or the considered arc. However, in the case of the K&G architecture, the parser uses fewer feature vectors than the models with no LSTMs. The transition-based parser (Figure 2.19a) selects only four feature vectors: $\vec{s_0}$, $\vec{s_1}$, $\vec{s_2}$, and $\vec{b_0}$. The graph-based parser (Figure 2.19b) uses only two representations: the head $\vec{h}$ and the dependent $\vec{d}$.

Extensions to K&G. Since the introduction of Kiperwasser and Goldberg's (2016b) architectures, BiLSTM-based parsers became a de facto standard in the parsing field. In the following years, several extensions have been proposed. The most known is the adaptation of Dozat and Manning (2017), who added *an attention layer* to the scoring procedure



(a) Transition-based parser scoring transitions for an example configuration $\langle [t_0, t_1, t_3], [t_5, t_6, t_7], \{\langle t_3, t_2\rangle, \langle t_3, t_4\rangle\}\rangle$.



(b) Graph-based parser scoring an arc $t_2 \rightarrow t_5$.

Figure 2.19: Schematic illustration of Kiperwasser and Goldberg's (2016b) architecture of Bi-LSTM-based neural dependency parsers.

of the graph-based architecture. The model achieved state-of-the-art performance and, after a few additional modifications described in detail by Dozat et al. (2017), its authors won the CoNLL 2017 Shared Task on Universal Dependency parsing (Zeman et al., 2017). In the consecutive year's shared task (Zeman et al., 2018), most of the best models, including the winning one by Che et al. (2018), used Dozat et al.'s (2017) architecture.

Among other modifications introduced by Dozat et al. (2017) were *character-based embeddings*. Such embeddings compose vector representations of words from the represen-

tations of their characters. The vector representations model morphological information better than basic word embeddings. They are especially well suited to parsing morphologically rich languages (Ballesteros et al., 2015). De Lhoneux et al. (2017) incorporated the embeddings into the transition-based K&G architecture.

Finally, Che et al. (2018), who were the winning team of the mentioned CoNLL 2018 UD Shared Task, improved the architecture's performance by adding *deep contextualized word embeddings*. Like the pre-trained word embeddings that we described earlier, such deep embeddings are learned from big amounts of unannotated text. However, during training, they do not focus on a fixed window size as the `word2vec` model. Instead, they use deep bidirectional architectures to incorporate the whole sentential context into the word representations. More specifically, such context can be learned from the internal states of LSTMs, as in the case of the ELMo model (Peters et al., 2018), or from transformers, as in the BERT representations (Devlin et al., 2019).

Feature extraction. At the end of this section it is important to notice that there is a clear advantage of BiLSTM-based parsing architectures over their predecessors: there is no need to design a complicated feature model. Kiperwasser and Goldberg (2016b) and Cross and Huang (2016) demonstrated that an architecture using only a few positional features (four for the arc-hybrid system and three for ARCSTANDARD) is sufficient to achieve state-of-the-art performance. Shi et al. (2017a) showed that this number could be further reduced to two features for arc-hybrid and arc-eager systems. A similar trend can be observed for graph-based dependency parsers. State-of-the-art models (Kiperwasser and Goldberg, 2016b; Dozat and Manning, 2017) typically use only two features of heads and dependents, possibly also incorporating their distance (Wang and Chang, 2016). Moreover, Wang and Chang (2016) show that arc-factored BiLSTM-based parsers can compete with conventional higher-order models in terms of accuracy.

2.3 Training Statistical Dependency Parsers

In the previous section, we described two parsing algorithms: transition-based and graph-based. The transition-based parser builds the final tree in a step-wise manner selecting the highest-scoring transition to apply at every step. The graph-based parser first scores all possible arcs (or factors) and then performs an exhaustive search to find the tree with the best overall score. In both cases, the parsing procedure is guided by a statistical model, i.e., weights used by the scoring function. These weights are obtained prior to

Algorithm 2.8 General online learning framework.

Input: Training data $T = \{\langle x_i, y_i \rangle\}_{i=1}^{N}$
Input: Number of epochs E
1: **function** TRAIN(T, E):
2: $w \leftarrow$ INIT()
3: **for** $epoch \in 1..E$ **do**
4: **for** $\langle x_i, y_i \rangle \in T$ **do**
5: $\hat{y}_i \leftarrow$ PREDICT(x_i, w)
6: $loss_i \leftarrow$ LOSS$(y_i, \hat{y}_i)$
7: **if** $loss_i \neq 0$ **then**
8: UPDATE$(w, x_i, \hat{y}_i, loss_i)$
9: **return** w

parsing through supervised learning, where the supervision comes from a gold-standard treebank of manually annotated trees.

General online learning. There are a plethora of options when it comes to supervised classifiers, and we refer the reader to Daumé (2015) for a comprehensive overview. The parsers used in this dissertation are trained with classifiers that follow the general *online learning* framework presented in Algorithm 2.8. The training procedure takes as an input training data $T = \{\langle x_i, y_i \rangle\}_{i=1}^{N}$ and a number of epochs E. It starts from initializing the weights w (line 2). Note that in the case of the traditional parsers, w is a vector. However, for the neural models, w denotes all the weight matrices and biases needed by the architecture. Moreover, the linear models initialize the weights with zeros, while the neural networks require randomly initialized parameters.

In the second step, the algorithm iterates E times through the whole training data. A common approach is to shuffle the training instances before every such iteration to make them more representative of the whole dataset. At every iteration, for each of the inputs x_i the algorithm makes a prediction $\hat{y}$ given the current weights w (line 5). Then, the prediction is compared to the correct output y, and the loss value is calculated (line 6). The loss measures compatibility between $\hat{y}$ and y. A common approach is to use the simple hinge loss, which aims at scoring the correct solution higher than the incorrect one with a margin m:

$$\text{LOSS}(y_i, \hat{y}_i) = max(0, m - (\text{SCORE}(y_i) - \text{SCORE}(\hat{y}_i))) \tag{2.15}$$

In the next step, the loss value is used to update the weights so that during the subsequent pass, the model makes a more accurate prediction (line 8). Finally, after all the epochs,

the weight vector (or matrices) is returned (line 9). Below we provide additional details to the described training procedure.

Training data and prediction. Transition- and graph-based dependency parsers use slightly different versions of the training algorithm presented above. Transition-based parsers are guided by multi-class classifiers. They are trained on oracle sequences of transitions, i.e., the training data consists of instances $\langle x_i, y_i \rangle$, where x_i are configurations, and y_i are correct transitions for these configurations. The classifier treats these transitions as classes: during the prediction step (line 5 in Algorithm 2.8) it iterates over all possible classes, scores them, and selects the highest-scoring one:

$$\text{PREDICT}(x_i, w) \leftarrow \arg\max_t \text{SCORE}(x_i, t, w) \tag{2.16}$$

In the case of graph-based parsers, the training data consists directly of sentences and gold-standard trees. This means that the size of the set of possible solutions $\mathcal{Y}(x_i)$ is exponentially large, and the outputs y_i cannot be simply iterated through. For this reason, graph-based parsers perform structured prediction (Collins, 2002), i.e., the prediction (line 5 in Algorithm 2.8) is performed with one of the graph-decoders described in Section 2.2.3.2, which can find the highest scoring tree efficiently:

$$\text{PREDICT}(x_i, w) \leftarrow \text{CLE}(x_i, w) \text{ or } \text{EISNER}(x_i, w) \tag{2.17}$$

UPDATE and traditional parsers. Details of the training procedure differ between traditional and neural parsers. The traditional parsers that we use are trained with the perceptron algorithm (Rosenblatt, 1958) extended by the passive-aggressive update and parameter averaging.

The basic perceptron algorithm increases the weights of the correct solution while decreasing the weights for the incorrect one. The passive-aggressive update extension (Crammer et al., 2006) introduces an additional scaling factor τ. This factor informs how to change the weights so that the current input is correctly classified after the update:

$$w_{i+1} \leftarrow w_i + \tau(\phi(x_i, y_i) - \phi(x_i, \hat{y}_i)), \tag{2.18}$$

where

$$\tau = \frac{w \cdot \phi(x_i, \hat{y}_i) - w \cdot \phi(x_i, y_i) + \delta(\hat{y}_i, y_i)}{||\phi(x_i, y_i) - \phi(x_i, \hat{y}_i)||^2}. \tag{2.19}$$

τ also ensures that the change will be big enough to classify the input by a margin equal to δ, but not bigger than needed. The margin δ depends on the task and the type of output. For multi-class classification, $\delta(\hat{y}_i, y_i)$ is set to 1, and for structured prediction, it is the number of tokens that received an incorrect head in the predicted tree $\hat{y}_i$.

The second extension to the basic perceptron algorithm is parameter averaging. In the online learning framework, updates are applied immediately after every training instance. As a result, the weight vector is more influenced by the very last instances than the ones processed earlier. Such sensitivity to the order of the training data can cause overfitting and thus decrease the generalizability of the models (Freund and Schapire, 1999; Collins, 2002). Collins (2002) tackled this issue and proposed an averaged perceptron. The method returns an average of all weight vectors seen during training and improves the generalization of the model.

UPDATE and neural parsers. Neural networks are trained using gradient-based methods. In general, the methods compute the gradient of the loss function with respect to each of the weights. This computation is performed with the backpropagation algorithm (Rumelhart et al., 1986; LeCun et al., 1998a), which allows for obtaining derivatives for complex networks efficiently. The computed gradient is used to update the weights – the update moves the weights in the opposite direction of the gradient and towards local optima. It is performed with one of many possible optimization algorithms, such as the Stochastic Gradient Descent (Bottou, 2012; LeCun et al., 1998b) or the Adam method (Kingma and Ba, 2014).

Training neural parsers can also differ from the traditional ones in terms of frequency of updates. In the online framework presented in Algorithm 2.8, an update is performed after every wrongly predicted output. However, such updates can also be performed after a few training instances or even the whole training data, which is commonly referred to as (mini)batch training. For example, Kiperwasser and Goldberg (2016b) summed local losses and performed updates once after every training sentence if the sum contained at least 50 elements.

2.4 Conclusion

In this background chapter, we gave an overview of the basic concepts that we rely on in the subsequent chapters. In particular, the parsing methods that we described serve as building blocks in the ensemble architectures studied in this dissertation. We provided references to related literature that describes these methods in more detail.

We started by providing the linguistic background that the parsing methods are based on. Particularly, we explained the notion of dependency structures and differences between projective and non-projective trees. Next, we looked into dependency treebanks that the data-driven parsers learn from. Especially, we focused on Universal Dependencies, which we use as our training data in most of the experiments and which facilitate the cross-lingual experiments in Chapter 5.

The next section covered parsing methods, i.e., algorithms that predict dependency trees for a given sentence. We started by looking into these methods from the perspective of the standard NLP pipeline to illustrate that dependency parsing is not an isolated task. It not only depends on other automatic tools, such as POS taggers but, more importantly, produces input for further downstream tasks, which directly depend on the parsing performance.

From the NLP pipeline, we moved to the two main paradigms of parsing: transition-based and graph-based. We started from the traditional methods developed before the neural methods were introduced into the parsing field. Later, we showed consecutive extensions that the deep learning approaches added to the traditional parsing architectures. Finally, we looked into how to train the statistical models from gold-standard data.

We would like to conclude this chapter by summarizing all the presented parsing methods from two general perspectives – parsing paradigms and differences between traditional architectures and neural models.

Transition-based vs. graph-based paradigm. From a machine-learning perspective, the two algorithmic paradigms present a trade-off between richness of feature representations on one hand and exact or approximate search methods on the other. Transition-based parsers provide rich access to structural features on partially predicted structures while relying on approximate search. Since the models greedily select the highest scoring transition, they may suffer from error propagation. However, the advantage of this approach is that it is fast and can work in linear time with respect to the length of the sentence. In contrast, graph-based parsers are limited in their scope of features in order to search the space of possible dependency trees exhaustively. As a result, they are guaranteed to find the tree with the highest score among all the valid dependency graphs. However, the decoders that they use are complex and require at least quadratic runtime.

Although the traditional transition- and graph-based paradigms have evolved in parallel, neither approach clearly outperformed the other in terms of accuracy. Especially, since the introduction of methods to tackle the error propagation problem in transition-based models, such as beam search (Zhang and Clark, 2008). Additionally, it was shown

that the two paradigms tend to make different types of mistakes that stem from their inherent differences (McDonald and Nivre, 2007). For example, transition-based parsers make more errors on long dependencies and these closer to the root, since decisions about them are usually performed in the last steps of parsing and, therefore, are influenced the most by error propagation. In contrast, the errors of graph-based models are more evenly distributed because their decoders do not prefer specific arcs. These differing error distributions and the complementary strengths of the parsing paradigms have given grounds to the integration methods that we discuss in the next chapter.

Traditional vs. neural parsers. In our overview of parsing methods, we focused on the neural extensions that lead to the architecture of Kiperwasser and Goldberg (2016b). However, we also believe that these are the most important deep learning additions to the parsing architectures introduced over the last few years. These additions can be summarized in three chronological steps.

First, the non-linear feed-forward networks replaced the linear models and reduced the need for complicated feature combinations. In the next step, dense representations allowed replacing the sparse binary indicator features with vector embeddings of words, POS tags, and labels. Finally, the introduction of contextualized representations allowed to encode the context of the whole sentence into the word representations. The last two techniques came with an additional advantage of incorporating external knowledge into the model by pre-training the representations on big amounts of unlabeled data.

Interestingly, these three extensions mostly deal with encoding transition-based configurations and graph-based factors. They successively allowed to add more information into feature representations while consecutively decreasing the manual effort needed to design a feature model. On the contrary, there was no change to the techniques for finding the final dependency tree after the configurations or factors are scored. The neural parsers involve the same transition systems and graph-based decoders as their traditional antecedents.

Chapter 3

Ensemble Dependency Parsers – Methods and Applications

The previous chapter presented the problem of dependency parsing and different methods to solve it. First, we defined the two main parsing paradigms: transition-based and graph-based. Then, we explained how these two approaches differ from each other and how both of them can be extended with deep learning techniques. Such traditional and neural single dependency parsers serve as building blocks in ensemble architectures throughout this dissertation, as illustrated in Figure 3.1.

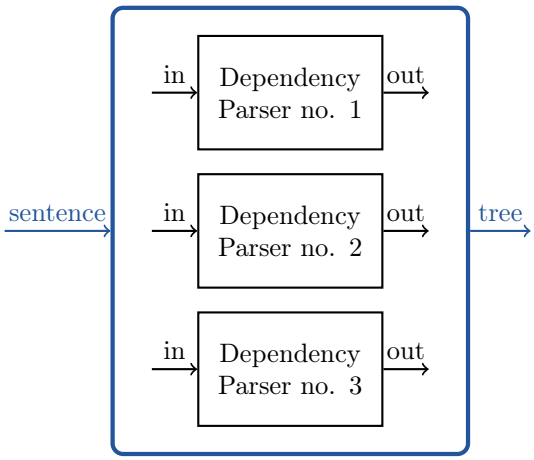


Figure 3.1: A general view on an ensemble dependency parsing architecture.

This chapter sets the scene for the following chapters by describing the ensemble dependency parsers. Such ensembles can differ in the procedure used to obtain the component models and/or the methods for combining their predictions. However, the parsing

literature shares the same three main assumptions about these approaches. First, the methods surely produce trees with higher quality than predictions of any of the single models. Secondly, ensemble approaches entail higher complexity. And finally, the approaches are "*not a practical solution*" (Kuncoro et al., 2016) and "*unusable by all but experts*" (Choi et al., 2015). Throughout this chapter, we explain the origins of these assumptions, analyze them across various methods, and study if they are correct.

The chapter starts by explaining the two ensemble approaches used in this thesis – feature-based stacking and blending. After addressing these two methods in separation, we review the similarities and differences between them. Then, we give an overview of other ensemble methods for integrating dependency parsers and outline how they differ from stacking and blending. Finally, we turn to the practical applications of ensemble dependency parsers and present possible use cases of such approaches.

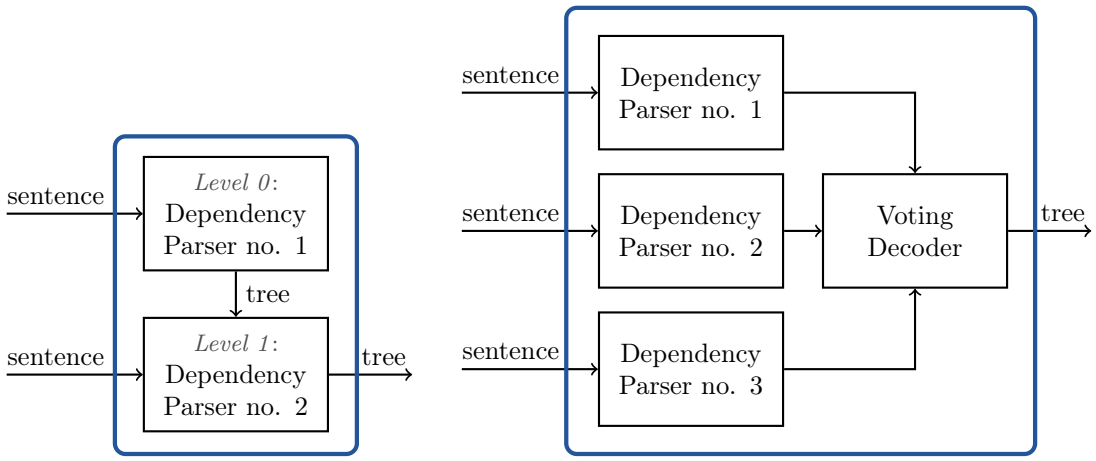
3.1 Ensemble Methods and Dependency Parsers

Ensemble methods allow to combine strengths of different classifiers and stabilize their performance (Dietterich, 2000). The techniques consist of performing the same task with few models and finding a consensus among their predictions. In this dissertation, we focus on applications of ensemble methods to the task of dependency parsing and refer the reader to Zhou (2012) for a general machine-learning perspective on ensemble algorithms.

In Natural Language Processing, system combination has been a common approach since the early years of the field. Ensemble methods have been applied to such tasks as POS tagging (Brill and Wu, 1998), partial parsing (Inui and Inui, 2000), or word sense disambiguation (Florian and Yarowsky, 2002). When it comes to dependency parsing, the combination approaches are usually divided into two categories: methods that integrate base parsers during *training* and during *prediction*. In this dissertation, we study one well-established representative from each of these categories: feature-based stacking and blending. We first describe both of these methods and then give a brief overview of other ensemble dependency architectures.

3.1.1 Combination during Training: Feature-based Stacking

Stacked learning or *stacked generalization* (Wolpert, 1992) is an ensemble method that involves two types of predictors: Level 0 and Level 1 (see Figure 3.2a for a stacking architecture with one Level 0 tool). The weak Level 0 models work in isolation. They are



(a) feature-based stacking **(b)** blending

Figure 3.2: A schematic view of the ensemble architectures studied in this dissertation.

trained in a standard way and make their predictions in separation. The Level 1 meta learner is trained on the predictions of all the base models. It learns how to combine them and correct their mistakes to make an improved decision.

Ensemble dependency parsers that use stacking typically integrate only two models, as presented in Figure 3.2a. Nivre and McDonald (2008) were the first to apply stacking to the task of dependency parsing. They showed that the method can take advantage of the complementary strengths of graph-based and transition-based parsers. Later, Martins et al. (2008) formalized the method and linked it to the concept of stacked learning. They also considered more combinations of parsers and experimented with first- and second-order graph-based models. In both of these publications, stacking was shown to lead to higher parsing accuracy than a non-stacked baseline. Moreover, in both cases, the ensemble methods achieved state-of-the-art results at the time of publication.

Feature models. *Feature-based stacking* involves running two parsers in sequence so that the second (Level 1) parser can use the output of the first (Level 0) parser *as features*. Nivre and McDonald (2008) defined a simple set of local features that the Level 1 parser extracts from the given predicted tree and adds to its basic feature model. In general, the features encode information about the arcs present in the input tree, and the model learns which of them it can trust.

Feature models depend on the type of the Level 1 parser. In the case of the transition-based parser, the feature function ϕ takes configurations as input. Therefore, the scope of feature templates must be limited to the stack, buffer, and the so-far predicted arcs. Ta-

Features for a predicted tree $\hat{A}$ and a configuration $c = \langle \sigma \vert s_1 \vert s_0, b_0 \vert \beta, A \rangle$
Is $\langle s_0, *, b_0 \rangle \in \hat{A}$?
Is $\langle b_0, *, s_0 \rangle \in \hat{A}$?
Head direction for s_0 in $\hat{A}$ (left/right/ROOT)
Head direction for b_0 in $\hat{A}$ (left/right/ROOT)
l' such that $\langle *, l', s_0 \rangle \in \hat{A}$
l' such that $\langle *, l', b_0 \rangle \in \hat{A}$

(a) Transition-based parser

Features for a predicted tree $\hat{A}$ and an arc $\langle h, l, d \rangle$
Is $\langle h, *, d \rangle \in \hat{A}$?
Is $\langle h, l, d \rangle \in \hat{A}$?
Is $\langle h, *, d \rangle \notin \hat{A}$?
Is $\langle h, l, d \rangle \notin \hat{A}$?
l' such that $\langle *, l', d \rangle \in \hat{A}$
l' such that $\langle h, l', d \rangle \in \hat{A}$

(b) Graph-based parser

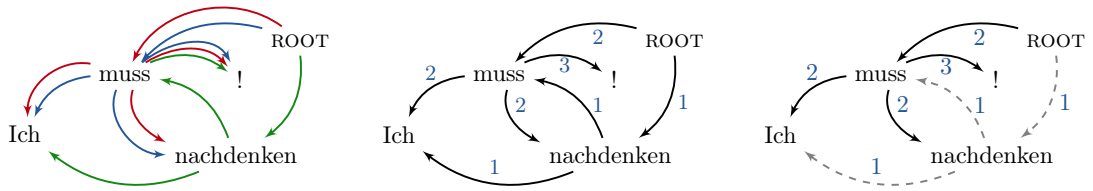
Table 3.1: Additional stacking features from Nivre and McDonald (2008). $*$ marks any label or node.

ble 3.1a presents Nivre and McDonald's (2008) feature model for transition-based parsers. It consists of six feature templates: the first two check if the potentially introduced arc between the front of the buffer b_0 and the first item of the stack s_0 exists also in the predicted tree $\hat{A}$; the next two check on which side (left, right, or ROOT) the heads of s_0 and b_0 are on; and the last two encode the syntactic relations of these two tokens in $\hat{A}$.

Table 3.1a presents corresponding features for the graph-based parsers. In this case, the feature function takes a single arc $\langle h, l, d \rangle$ as input. The feature model adds six feature templates with stacking information: the first four check if the Level 0 parser also predicted the arc (with or without the label l), the last two encode the syntactic relations of the head token h and the dependent d in $\hat{A}$.

Martins et al. (2008) extended this feature set to include non-local information, e.g., information about dependents' siblings and grandparents. They found that the additional non-local features provide only further minor gains over the local ones if the Level 1 parser itself already uses non-local features. Based on that, they interpreted stacking as a way of approximating rich non-local features in models.

Training. Finally, it is important to add that training stacking models has to be performed in two steps. In the first step, the training data is annotated by the Level 0 parsers under a cross-validation scheme, usually referred to as *N-fold jackknifing*. The procedure first splits the training treebank into N partitions. Then, N instances of Level 0 parsers are trained in a leave-one-manner: $N - 1$ partitions are used as training data and the remaining partition as the test data. In the second step, the Level 1 parser is trained on the whole training data additionally annotated with the Level 0 trees.



(a) predictions from three parsers **(b)** votes **(c)** merged tree

Figure 3.3: Three steps of blending for an example sentence "*Ich muss nachdenken!*" (eng. I have to think!).

3.1.2 Combination during Prediction: Blending

Voting is an ensemble method that consists of running a set of classifiers in separation. The method classifies new data points by taking a (weighted) vote of all the single predictions.

Zeman and Žabokrtský (2005) were the first to apply the voting schema to the dependency parsing task. Later, different modifications of this method were referred to as *blending* (Hall et al., 2007) or *re-parsing* (Sagae and Lavie, 2006). Figure 3.3 presents an application of blending to a German example sentence "*Ich muss nachdenken!*" (eng. I have to think!).[1] In general, blending consists of three steps: (a) running a couple of parsers in separation and combining their outputs into one graph; (b) calculating the scores for arcs in the combined graph; (c) finding the final tree. Since the first step is rather straightforward, we provide more details for the other two below.

Voting and weights. The second step of blending consists of mapping the predictions of single models into arc scores. Various methods have been proposed to perform this mapping. The basic approach presented in Figure 3.3b simply counts how many basic models predicted a particular arc and takes this vote as the arc score. More complicated methods weight arcs depending on parsers that predicted them. However, it depends on the application if such additional weights provide accuracy gains.

Zeman and Žabokrtský (2005) proposed to take into consideration the accuracy of the single models so that the better performing parser has a bigger influence on the final result. Considering that parsers specialize in different phenomena, Sagae and Lavie (2006) demonstrated that further improvements can be obtained by making the weights related to the dependents' POS tags. However, later Surdeanu and Manning (2010) found

[1] This figure is based on my contribution to the publication (Schweitzer et al., 2018).

the differences between these two models to be rather marginal. A similar pattern was observed by Fishel and Nivre (2009), who analyzed additional phenomena such as dependency relation or the POS tag of the head and observed only modest improvements.

Decoding. Finding the best tree for the combined weighted graph is the final step of blending (see Figure 3.3c). Zeman and Žabokrtský (2005) used a simple head selection, where each of the words gets assigned the head with the highest weight. However, such an approach does not guarantee that the resulting structure will be a well-formed and connected dependency tree. For that reason, Sagae and Lavie (2006) proposed to use a graph decoder, specifically Eisner's algorithm, to find the maximum spanning tree in the combined graph (we refer to Section 2.2.3.2 for the explanation of graph-based decoders). Later, this approach became a de facto standard way to perform combination of dependency parsers during prediction.

The graph-decoder causes the time complexity of blending to be at least quadratic (for the Chu-Liu-Edmonds decoder) or cubic (Eisner's algorithm), even when the basic models run in linear time. As an alternative, Attardi and Dell'Orletta (2009) proposed an approximate combination method that runs in linear time. The method builds the final tree in a top-down way. It starts from the ROOT and, in a step-wise manner, expands the tree by selecting arcs with the highest score. Later, Surdeanu and Manning (2010) compared these two approaches and found out that the empirical difference between them is negligible, especially on the out-of-domain data, which makes the approximate combination a good choice for practical applications. Finally, Fernandez-Gonzalez et al. (2016) showed an alternative approach which blends only two parsers: a left-to-right transition-based model, which decides about the tokens at the beginning of the sentence, and a right-to-left one, which makes decisions about the ones at the end. However, in contrast to all the above-mentioned approaches, this method is restricted only to transition-based parsers.

3.1.3 Feature-based Stacking vs. Blending

Before we turn to the other ensemble parsing architectures, we give a brief comparison of the two methods studied in this dissertation. With respect to the final performance of the ensemble, both Fishel and Nivre (2009) and Surdeanu and Manning (2010) compared the methods and found that blending outperforms stacking. However, the method comes with other advantages listed below.

Differences. The main difference between stacking and blending is the way the single models communicate with each other. In stacking, the Level 1 parser is aware of the errors of the Level 0 model and can learn how to improve them. Moreover, the method needs only two single models to perform a successful combination.

When it comes to blending, the models do not communicate with each other, and the final decoder is not able to learn from the previous mistakes. The advantage of such an approach is that the method does not require additional effort at the training time, such as jackknifing the training data. However, the method needs at least three single models for the voting scheme to work correctly, which is an additional cost comparing to stacking. But, the ability to take into consideration more predictions is also the main advantage of blending. Surdeanu and Manning (2010) showed that the more parsers are involved, the better the performance of the ensemble.

Similarities. In general, ensemble methods lead to better results when they employ uncorrelated models (Breiman, 1996; Kuncheva and Whitaker, 2003). The intuition is that when the models are similar, they produce similar errors, and these errors get more weight (for example, votes) than other predictions.

The main similarity between stacking and blending is, therefore, their dependence on the *diversity of the involved models*. In the case of stacking, Surdeanu and Manning (2010) presented a study on English and found that the diversity of the involved parsing algorithms is essential. Specifically, stacking a parser on itself does not lead to accuracy gains. This effect was also observed by Martins et al. (2008), who found that integrating two parsers of the same type gives at most minor improvements. However, such diversity does not have to come from differences in transition- and graph-based parsing paradigms. For example, Attardi and Dell'Orletta (2009) proposed to combine two transition-based parsers that process the input left-to-right and right-to-left. The method improved the performance of the single models while maintaining an overall linear complexity of the whole architecture.

When it comes to blending, the method is usually applied to a mixture of graph-based and transition-based left-to-right and right-to-left parsers. The first application of Zeman and Žabokrtský (2005) was a mixture of seven methods, including also a rule-based system. The authors showed that even the weakest parsers contribute to the overall improvement coming from the ensemble. Among the later applications, Sagae and Lavie (2006) combined three transition-based models and one graph-based model. Hall et al. (2007) achieved the best performance at the CONLL 2007 shared task (Nivre et al., 2007) with a mixture of six left-to-right and right-to-left transition-based parsers with different

transition systems.

3.1.4 Other Ensemble Approaches

Feature-based stacking and blending were the first ensemble methods proposed for dependency parsing, but not the only ones. We now provide an overview of other combination methods with the primary goal of outlining in which way these methods differ from stacking and blending. We focus on methods that integrate at least two dependency parsers. Therefore, methods that combine a dependency parser with parsers for other syntax frameworks, such as Lexical Functional Grammar (LFG) (Øvrelid et al., 2009a,b) or phrase-based structures (Farkas and Bohnet, 2012), are out of the scope of this overview.

Integration through features. The question of how to perform a more integrated combination of the strengths of transition- and graph-based parsers was addressed by Zhang and Clark (2008) and Bohnet and Kuhn (2012). The authors proposed a training-time integration method that employs features coming from both of the paradigms. Specifically, they introduced beam search-based transition-based parsers with features strongly inspired by graph-based models. The main difference between their approaches was that Bohnet and Kuhn (2012) additionally re-scored elements of the beam with the graph-based completion model, while the transition-based parser gradually built partial structures. Compared to feature-based stacking , the approach does not need two predictors running at the parsing time or jackknifing the training data. However, it comes with the requirement that the final decoder is a transition-based one.

Integration through data. One of the main bottlenecks of statistical dependency parsers is the dependence on manually-annotated training sentences. A common way to tackle the problem of insufficient training data is weak supervision. The approach, in contrast to the methods studied in this dissertation, makes use of additional, unlabeled data. It involves running parsers in two steps: first, parsing unlabeled sentences with one parser, and then, using the obtained predictions to train another model. The underlying idea is that although the predictions of the first model are imperfect, they still contain a lot of correctly predicted dependencies. And the second model will be able to learn from them more than only from a small, manually-annotated treebank. Depending on the number of involved models, this method is referred to as *self-training* (Yarowsky, 1995), *co-training* (Blum and Mitchell, 1998), or *tri-training* (Zhou and Li, 2005). All the three variants have

been applied to dependency parsing with varying success (see Spreyer and Kuhn (2009) for self-training, Sagae and Tsujii (2007) for co-training, and Søgaard and Rishøj (2010) for tri-training).

Integration through stacking. In general, the concept of stacking refers to running two types of predictors: weak Level 0 models and strong Level 1 classifiers, which fix the errors of the weaker tools. In feature-based stacking, the Level 1 predictor is a parser. However, it can be any statistical classifier that can learn which Level 0 outputs to trust. In the parsing field, modifications of this approach were first applied to the constituency parsing and referred to as *parse hybridization* (Henderson and Brill, 1999) or *re-ranking* (Collins and Koo, 2005; Charniak and Johnson, 2005; Hall, 2007).

Zeman and Žabokrtský (2005) were the first to apply the general stacking approach to dependency parsing. They used seven different Level 0 models and a Level 1 classifier to decide which of them to select. Later, Søgaard and Rishøj (2010) went one step further. They first applied feature-based stacking and then a Level 2 classifier to decide among predictions of the stacked models. In both of these approaches, Level 0 predictions came from different algorithms. A different method, usually referred to as re-ranking, involves predicting multiple outputs with the same Level 0 model and selecting one of them with a more robust Level 1 algorithm. Such an approach allows for limiting the number of possible solutions before applying a model with rich, high-order features (Sangati et al., 2009; Hayashi et al., 2013, among others).

All the above-described approaches differ from blending in two ways. First, the Level 1 tools are statistical classifiers which have to be trained before application. Secondly, these tools decide between full trees and select one of them. In contrast, blending applies the voting scheme, which does not need training. Moreover, it can create an entirely new tree, which was not present among the predictions of the models.

Neural-specific integration. Since the introduction of deep learning techniques, ensemble methods have received relatively little attention in the parsing literature. One of the possible explanations might be that since the neural parsers are constantly advancing state-of-the-art in the field, improving their performance even further with ensembles became less attractive. However, a several approaches can be found, especially related to stabilizing the performance of the neural models which are sensitive to initialization (Reimers and Gurevych, 2017). For example, out of 24 teams participating in the CoNLL 2018 shared task on dependency parsing (Zeman et al., 2018), five employed ensemble techniques. All of them took advantage of either diversity coming from random seeds

or different languages, and not from different parsing paradigms. Such combining neural models of the same type can be performed by taking the sum of their MLP scores (Che et al., 2017), averaging softmax scores (Che et al., 2018), or simply through blending (Kuncoro et al., 2016).

3.2 Ensemble Dependency Parsers and Applications

The previous section summarized different approaches for combining parsing architectures. Most (if not all) of the cited publications introduced ensemble methods as tools for improving the performance of single models, usually by benefiting from the diversity among parsing methods. However, boosting the accuracy of parsers is only one of many possible use cases for ensemble methods in dependency parsing. Therefore, in this section, we take a closer look at other useful applications.

While the goal of the previous section was to give a broad overview of different ensemble parsers, we now focus on concrete examples of practical applications of such methods. The examples can be arranged into four groups: (1) combining strengths of parsing architectures; (2) cross-lingual methods; (3) improving downstream tasks; and (4) resource preparation. Since Chapter 5 focuses entirely on cross-lingual methods, we leave this use case out of this overview and focus on the other three groups.

3.2.1 Combining Strengths of Parsing Architectures

Before we turn to the less common applications of ensembles, we stay in the context of improving the performance of single models and explain *why the ensemble methods are most commonly associated with maximizing parsing scores*.

Shared tasks and parsing. Historically, one of the milestones for dependency parsing were the two CoNLL shared tasks focusing on multilingual dependency parsing (Buchholz and Marsi, 2006; Nivre et al., 2007). They not only resulted in state-of-the-art methods but also set the scene for the following years. For example, they established a methodology of testing parsing algorithms, such as dependency treebanks and evaluation methods. The two competitions were followed by many other shared tasks related to dependency parsing, focusing on joint parsing of syntactic and semantic dependencies (Surdeanu et al., 2008; Hajič et al., 2009), parsing web texts (Petrov and McDonald, 2012), and morphologically rich languages (Seddah et al., 2013, 2014). The more recent CoNLL

shared tasks took place in 2017 and 2018 (Zeman et al., 2017, 2018), and once again focused on multilingual dependency parsing. Finally, the most recent IWPT 2020 shared task involved parsing into enhanced Universal Dependencies (Bouma et al., 2020).

Evidently, shared tasks have a long tradition in the parsing field. Most of the winning teams of these competitions used ensemble approaches (Hall et al., 2007; Björkelund et al., 2013; Che et al., 2018, among others). Moreover, an interesting pattern can be observed when analyzing the winning submissions over the years. The two teams with the highest scores in the first CoNLL 2006 shared task, McDonald et al. (2006) and Nivre et al. (2006a), used a single graph-based and a single transition-based parser, respectively. In the next year, Hall et al. (2007) won the CONLL 2007 shared task by employing an ensemble of six parsers. Almost ten years later, the first shared task after the introduction of deep learning methods into the field, was won by Dozat et al. (2017). Their submission involved only a single neural graph-based parser and surpassed strong ensembles of traditional, non-neural parsers. Therefore, already the next year's winning team employed an ensemble of three Dozat et al.'s (2017) parsers (Che et al., 2018).

Multilingual solution. A clear pattern emerges from the history of parsing shared tasks – *over the years, ensemble methods have been the go-to approach to determine the state-of-the-art in the parsing field*. Moreover, the ensembles were providing an additional benefit that is commonly overlooked. All of the above-mentioned shared tasks involved parsing multiple datasets, from 13 languages in the case of the first CoNLL 2006, to 82 treebanks in 57 languages in CoNLL 2018. In such a highly-multilingual setting, ensembles serve as *language-agnostic architectures*. In other words, combining multiple parsers with different language-specific strengths offers robust results without the need to select one model for each of the use cases.

The multilingual benefit of ensembles can be illustrated through the results of our submission to the CONLL 2017 shared task (Zeman et al., 2017). Our architecture, described in details in (Björkelund et al., 2017), involved three parsers: a traditional transition-based parser (TP), a traditional graph-based parser (GP), and a neural transition-based parser with character-based embeddings (TN). We trained multiple instances of each of these parsers, changing the random seeds and (for the transition-based models) the direction of processing the input sentence. The final submission was an ensemble architecture that combined the single models through blending.

Figure 3.4 presents the results of the three baseline single models and the ensemble on

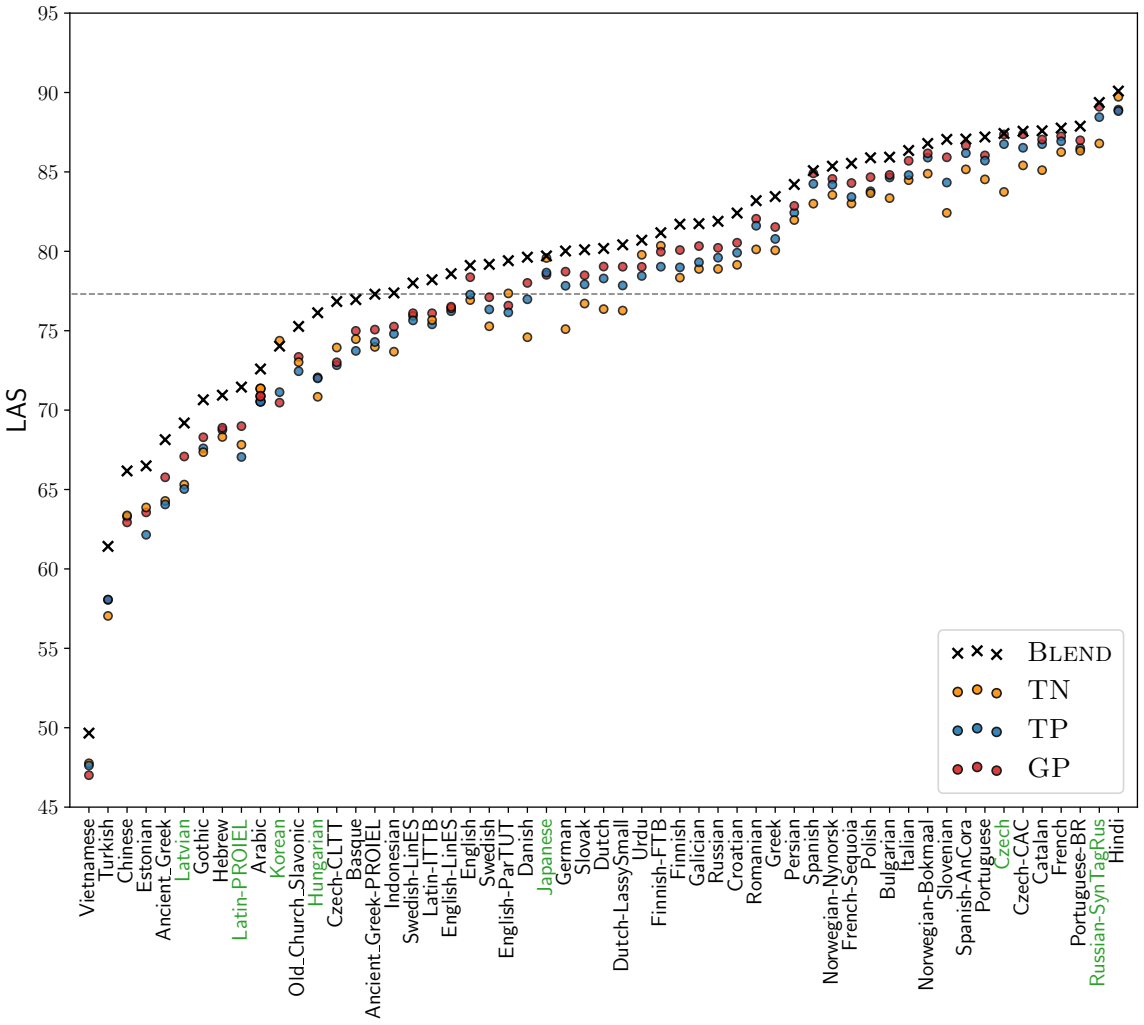


Figure 3.4: Development results of our submission to the CONLL 2017 shared task (Zeman et al., 2017). Treebanks that we refer to in the text are marked in green. The horizontal dashed line marks the average performance of all single models (77.31 LAS).

the 55 development treebanks from the CONLL 2017 shared task.[2] The parsing performance is evaluated with Labeled Attachment Score (LAS), i.e., the ratio of tokens with a correct head and label to the total number of tokens. The average across all single models (77.31 LAS) is marked with a horizontal dashed line.

We can observe that none of the three baseline parsers performs the best across all the treebanks. Contrarily, each of them has an advantage in particular use cases. For example, TN, which uses character-based embeddings, has a strong advantage on the

[2]We refer to Appendix A.2 for the treebank statistics.

languages in which characters play an important role, such as Korean or Japanese (we mark in green languages that we refer to). The traditional parsers behave in a more similar way, however, the difference between them still varies a lot depending on the treebank, reaching up to 2 LAS points for Latvian and 1.9 LAS for Latin-PROIEL. Finally, for almost all treebanks blending improves over the best single model (the exception is Korean, for which TN achieves 0.34 LAS more than blending). Moreover, a general trend can be noticed: the improvements are stronger for the treebanks which are harder to parse (left of the figure, accuracy below the average threshold) and smaller for the treebanks with big amounts of training data, such as Russian-SynTagRus or Czech. The biggest improvement of 4 LAS points comes for Hungarian, which is one of the most challenging treebanks. Its training treebank is second to the last in size (after Czech_CLTT) and its best single model achieves only 72 LAS compared to an average 77.3 LAS across all treebanks and models.

To sum up, we used the results from CONLL 2017 shared task to illustrate that different baseline parsers have different accuracy depending on the treebank they parse. Blending levels the performance of diverse models regardless of the language and domain of application. In other words, employment of an ensemble method allows for applying the same architecture across all the treebanks and achieving robust results without language-specific model selection.

3.2.2 Improving Downstream Tasks

In practical NLP applications, parsers are rarely the final step of linguistic analysis. More commonly, they produce input for further processing tasks. We now take a closer look at two concrete examples of such downstream tasks – enhanced parsing and machine translation – and the influence that ensemble dependency parsers can have on their performance.

Enhanced parsing.[3] Enhanced parsing is another example of semantic parsing tasks that we saw in the NLP pipeline in Figure 2.5. It involves predicting representations that extend the basic Universal Dependencies with auxiliary relations, such as dependencies between content words. Such enhanced graph representations can be more beneficial for downstream tasks than the basic UD trees (Oepen et al., 2017).

[3]This paragraph is based solely on my contribution to the publication (Falenska et al., 2018).

Predicting enhanced graphs can be approached in two ways: (1) either by directly parsing to the final graph-based enhanced representations; (2) or in a two-stage process, in which surface syntax trees are predicted first and then enhanced with additional arcs. Schuster et al. (2017) analyzed both these approaches and showed that the second one leads to better downstream performance within their setting. However, the approach is a pipeline and comes with such drawbacks as error propagation. That is, the lower the quality of parsed trees, the worse the final enhanced representations.

In Falenska et al. (2018), we studied such a pipeline approach and analyzed if reducing errors of parsers through ensemble methods translates into improvements in enhanced parsing. More specifically, we performed experiments on the Polish treebank PDBUD annotated with enhanced representations (Wróblewska, 2018). Our approach consisted of three steps. First, sentences were parsed with a variety of parsing models. Then, the single outputs were combined into one graph through blending. And finally, the blended tree was augmented with enhanced relations through 12 simple rules. The rules were designed manually and guided by the intuition of a Polish native speaker.

We found that rule-based augmentation strongly depends on the employed parsing system. If perfect parsing is assumed (by using gold-standard trees), the 12 simple rules can achieve extremely high accuracy, leaving little space for further improvements. However, since the rules are not built to handle parsing errors, the parsing accuracy directly translates into performance on predicting the enhanced arcs.

Machine translation. While enhanced parsing, similarly to dependency parsing, is rather one of the intermediate steps in the NLP pipeline, machine translation is a concrete final downstream task. Moreover, syntax-based machine translation is a typical example of an application that builds upon the quality of the syntactic analysis.

Green and Žabokrtský (2016) provided a comprehensive analysis of how the quality of dependency trees influences machine translation, specifically, the TectoMT English-Czech translation system (Popel and Žabokrtskỳ, 2010). Through a variety of parsing techniques and different ensemble approaches, e.g., weighted blending and stacking with a Level 1 classifier, they found a strong correlation between the parsing performance and machine translation scores. Moreover, in their results, the most significant improvements in translation stemmed from using combined parsing models. However, similarly to the enhanced parsing example, using gold-standard trees as the input to the downstream task revealed much space for further improvements.

Task-oriented combination. Contrary to what we have seen in the two examples above, an improvement in parsing accuracy does not always translate to an improvement in the downstream performance. For example, Gómez-Rodríguez et al. (2017) evaluated four single dependency parsers on the task of predicting the polarity of sentences based on their parse trees. They found that using better models does not necessarily imply a better polarity classification. Similarly, Katz-Brown et al. (2011) investigated a self-training setting in which a parser is trained not to achieve the best possible accuracy but towards an extrinsic evaluation metric. In their evaluation, worse overall parsers gave better subjective quality of machine translation.

Eckart (2018) in her dissertation analyzed this effect and emphasized that only specific linguistic phenomena are relevant for specific downstream tasks. She proposed a full infrastructure for a parser combination that can take the downstream task's specificity into account. The infrastructure allows for combining different syntactic frameworks, not only dependency trees, and enhance the reliability of automatically generated outputs.

To summarize, we saw two examples of tasks that use syntactic analysis as input – enhanced parsing and syntax-based machine translation. In both of the cases, *improvements coming from ensembles directly translated to better downstream performance*. However, for tasks that depend on specific linguistic phenomena, such mapping might not be as straightforward. In such cases, ensemble architectures that involve the specificity of the downstream task, such as re-ranking or blending with task-specific weights, *allow for selecting the solution that fits the best to the particular application*.

3.2.3 Resource Creation

Gold-standard resources, i.e., manually curated annotated corpora, play an important role both in the language sciences (linguistics, psycholinguistics) and in speech and language technology. They serve as data for testing hypotheses, evaluating automatic systems, and providing the signal in supervised training of machine learning models.

The creation of gold-standard resources requires the involvement of humans and is, therefore, time-consuming and costly. Moreover, the higher the complexity of the annotations, the more attention of the annotators is needed, and the slower the whole process. For example, while POS annotation can be relatively fast (Garrette and Baldridge, 2013), creating treebanks is more problematic (Zeman and Resnik, 2008; Souček et al., 2013). Therefore, methods that allow reducing the manual workload in the treebank creation are desired. We now look into examples of such methods that use ensemble dependency

techniques.

Semi-automatic annotation. Fully-manual annotation requires the annotators to process each of the sentences from scratch. However, some of the dependencies, like attaching articles to nouns, are less ambiguous than others and could easily be predicted by automatic tools. Such a semi-automatic process involves two steps: the sentences are first parsed and then checked by humans, and if necessary, corrected. Such an approach was taken, for example, during the creation of the first universally annotated treebanks (McDonald et al., 2013) or the Polish Składnica treebank (Świdziński and Woliński, 2010).

In the *semi-automatic process*, the better the predicted annotations, the less effort is needed from the annotator to correct them. In such cases applying several single parsers and an ensemble combination might directly reduce the manual workload. Moreover, new treebanks are commonly developed for new types of texts and/or domains. In such cases, parsers are applied in the *out-of-domain setting*, i.e., they are trained on large gold-standard canonical data and applied to non-standard texts. Even the state-of-the-art parsers perform worse when applied to unknown text genre.[4] Therefore, since ensemble methods have been shown to improve the models also out-of-domain (Surdeanu and Manning, 2010), they can directly speed up the annotation process by overcoming the performance drops related to the out-of-domain applications.

Sohl and Zinsmeister (2018) studied manual syntactic annotation of texts coming from German books for secondary schools. By analyzing the performance and errors made by three parsers they estimated to what extent an ensemble can support the process of manual annotation. They found that the parsers strongly agree on certain labels, such as determiners, and completely agree on 15% of sentences. They concluded that in such cases the annotators could entirely skip processing of such words or sentences.

Partial-trees. The idea of manually processing only difficult cases is strongly related to the concept of *active learning*, in which the instances to annotate and later train on are selected algorithmically (Hwa, 2004). In the context of syntactic annotations, such partially-annotated trees can be filled in automatically (Mirroshandel and Nasr, 2011) or directly serve as training data (Flannery and Mori, 2015). In both these cases, the step of selecting which arcs to annotate manually is crucial. For example, Flannery and Mori (2015) showed that by effectively selecting meaningful training instances, the amount of data needed for domain adaptation of a dependency parser can be reduced up to 75%.

[4] In general, machine learning has a relatively strong dependency on the text genre (Sekine, 1997).

The information about the problematic parts of the sentence can come from n-best predictions of one parser (Hwa, 2004; Mirroshandel and Nasr, 2011; Flannery and Mori, 2015) or from multiple models. In the latter, disagreements between different parsers can serve as a very meaningful confidence measure (Surdeanu and Manning, 2010).

Silver-standard resources.[5] The required human involvement is the bottleneck in the creation of gold-standard treebanks. The silver-standard approach is an alternative to the gold-standard concept. It is based on an observation that not all resources can be fully annotated manually, for example, due to their size or non-static nature. In such situations, it might be necessary to loosen the restrictions of the gold-standard approach and provide annotations with lower quality. Such annotations might facilitate research for new domains or allow for combining research branches.

The *silver-standard resources* implicitly employ the ensemble approach. More specifically, to provide the best-quality automatic predictions, the resources come with annotations that are a result of combining and harmonizing outputs of different automatic tools. The main expectation is that the quality of annotation exceeds the performance of any of the single systems and narrows the gap to manually annotated resources. The approach was proposed by Rebholz-Schuhmann et al. (2010), who released the CALBC – a corpus of biomedical named entities annotated automatically by a selection of independently developed systems.

The German RAdio INterviews corpus (GRAIN, Schweitzer et al. (2018)), part of the SFB732 Silver Standard Collection, is another example of resources following the silver-standard idea. It consists of 140 German radio interviews annotated on multiple linguistic layers. The corpus is built from two parts: a gold-standard part with manual annotations for a subset of 20 interviews, and a (much larger) silver-standard part, relying entirely on automatically predicted information.

The silver-standard part of the corpus consists of over ten layers of automatic speech and text annotations, including syntactic analysis. The analysis was predicted by five tools, among which four provided dependency trees. The predicted dependency trees were additionally post-processed in two ways. First, they were converted into confidence estimations based on the agreement between different systems. While this extra step does not directly increase the annotation quality, it provides valuable information about the relative reliability of annotations. Second, the separate dependency trees were combined through blending, and the resulted merged trees were provided as an addi-

[5]This paragraph is based on my contributions to Sections 1 and 3 from the publication (Falenska et al., 2020b).

tional layer of annotation. This step was based on an assumption that enriching a treebank with information about combined trees improves its usability for users who prefer to work with only a single dependency tree instead of multiple predicted ones. Moreover, it increases the reliability of the provided trees, since ensemble parsers are known for achieving higher performance than single models.

To summarize, we have discussed the creation of gold-standard, manually curated resources. We noted that the bottleneck in the development of such resources comes from the involvement of humans. We saw two methods to decrease the human workload: (1) semi-automatic annotation approach, in which *ensembles can provide robust annotations to start from*, and (2) active-learning, where *predictions from multiple parsers can serve as measures of confidence*. Finally, we discussed resources for which manual annotation is not feasible and the silver-standard approach. We explained that the silver-standard treebanks by design come with multiple predicted trees, and their *usability can be improved* by providing an additional layer of merged trees.

3.3 Conclusion

In this chapter, we studied ensemble dependency parsers. We started by outlining the three assumptions about these parsers that are common in the literature. We would like to conclude this chapter by addressing these assumptions and looking at the ensembles from the perspectives of accuracy, complexity, and usability.

Accuracy. In general, ensemble methods are known for achieving more accurate and robust results than single models (Dietterich, 2000). NLP tools that employ ensembles, such as ensemble dependency parsers, inherit this property. Over the years, the ensemble parsers have proven to be successful in a variety of parsing shared tasks. They brought gains in parsing accuracy across typologically diverse languages, in- and out-of-domain applications, and in downstream tasks. Moreover, they served as multilingual tools allowing for achieving robust results without language-specific model selection.

However, when it comes to the accuracy of ensembles, one important factor has to be remembered. The gains coming from the integration directly depend on *the diversity among the combined architectures*. Therefore, combining predictions with the same error distributions is not beneficial.

Complexity. Ensembles are attractive since they improve the performance of single baseline models. They pay for these improvements by being more complicated and challenging to maintain. They require training and running multiple models and designing the way these models communicate with each other. However, the asymptotic runtime of ensembles is *not necessarily higher than the asymptotic runtime of the models involved*.

The runtime of the whole ensemble is determined by the slowest decoder in the architecture. When it comes to feature-based stacking, the method requires parsing sentences with two single models. Since these models have to run in sequence, their combined runtime is the sum of the individual models. However, the overall complexity can still stay linear, as we saw in the case of integrating two transition-based parsers processing the input from different sides (Attardi and Dell'Orletta, 2009).

Blending involves at least three diverse models. Since these models do not depend on each other, they can run in parallel. Therefore, the complexity of the whole ensemble depends on the slowest single model and the final voting decoder. We saw architectures in which this decoder increased the overall complexity class to quadratic or even cubic with respect to the length of the sentence. However, we also described an approximate algorithm of Attardi and Dell'Orletta (2009), which achieved empirically comparable results in linear time.

Usability. The long tradition of parsing shared tasks might be the reason why the ensembles are mostly associated with boosting parsing accuracy. Moreover, due to their additional processing cost, they might seem complex. As a result, research publications about practical applications of ensemble parsers are not common. However, in the examples that we described, one reappearing pattern becomes clear – combining different parsing methods does not only bring parsing improvements but *is a powerful tool for receiving robust and reliable results*.

Especially when it comes to practical applications, end users are often facing the dilemma which of the many available off-the-shelf parsers to apply. Such a decision has to take into consideration a variety of methodological aspects, such as the interrelations between the parsing algorithms and the particularities of the language to parse. This choice is especially difficult when there is no gold-standard data to evaluate the models on, such as in the out-of-domain applications. In such scenarios, ensemble parsers allow the end users to have even *less expertise*. Instead of selecting one tool they can combine multiple parsers and achieve a quality that exceeds the performance of any single system. Moreover, since the ensembles come with multiple outputs, they bring the additional notion of confidence, which allows for analyzing which predictions are error-prone and

should not be trusted.

Chapter 4

Supertagging vs. Stacking for Dependency Parsing

In this chapter, we look at ensemble dependency parsers from the first methodological perspective addressed in this dissertation: **parsing time**.[1]

Ensemble architectures involve running multiple parsing algorithms. Therefore, they come with higher processing costs than a single parser. Especially troublesome from this perspective are methods that combine parsers during training (see Chapter 3, Section 3.1.1). Such methods, contrary to the approaches that combine models during prediction, do not allow for reducing the processing time by running single models in parallel. For example, in Chapter 3 (Section 3.1.1), we discussed the integration method called *feature-based stacking*. Recall that the method requires two single parsers: one Level 0 parser and one Level 1. These parsers have to predict trees in sequence so that the Level 1 model can use the output of the Level 0 model as features. Therefore, the runtime of the individual models adds up and not parallelizes, and the slowest decoder in the architecture determines the asymptotic complexity of the whole ensemble.

The first stacking architecture of Nivre and McDonald (2008) involved a graph-based and a transition-based parser. The complexity of such an architecture is at least *quadratic* and stems from the complexity of the graph-based decoder. A straightforward approach to design a faster ensemble is to build it from faster single models. For example, in the previous chapter, we discussed the architecture of Attardi and Dell'Orletta (2009), which combines two greedy transition-based parsers processing sentences from left-to-right and right-to-left. Replacing the graph-based parser with a reversed transition-based model

[1]The content of this chapter is based on (Faleńska et al., 2015).

keeps the asymptotic complexity of the ensemble *linear*.

We continue the line of research initiated by Attardi and Dell'Orletta (2009) and investigate how to decrease the parsing time of feature-based stacking. However, we approach this issue from a different perspective and ask *if the involved models have to predict well-formed dependency trees.* Our hypothesis is that the Level 0 parser in feature-based stacking can be replaced with a fast *supertagger* without losing the effectiveness of the ensemble. Before we motivate this hypothesis, we explain the concept of *supertagging*, which has not been addressed in this dissertation yet.

Supertagging. Supertags are labels for tokens that encode syntactic information, e.g., the head direction or the subcategorization frame. The term *supertag* has been introduced by Joshi and Bangalore (1994) as an elementary structure associated with a lexical item. These elementary structures carry more information than POS tags, hence the name super POS tags or *supertags*.

Supertagging was originally developed for rich grammar formalisms as the disambiguation of the supertag assignment *prior to parsing*. Within the Lexicalized Tree Adjoining Grammar (LTAG) (Schabes et al., 1988), supertags correspond to trees that localize dependencies. First, a supertagger assigns supertags to each word of a sentence, and then a parser combines these structures into a full parse. Such a combination leads to simplified and faster parsing (Bangalore and Joshi, 1999). The same approach was successfully applied to the Combinatory Categorial Grammar (CCG) (Clark and Curran, 2004) and the Head-Driven Phrase Structure Grammar (HPSG) (Ninomiya et al., 2006). In both cases it resulted in significant speed improvements.

Foth et al. (2006) were the first to use supertags in a dependency parsing context. Specifically, they incorporated them as soft constraints into their rule-based parser (Foth et al., 2004). In LTAG, CCG, or HPSG, supertags are the elementary components of the framework in question. In (Foth et al., 2006), supertags were explicitly designed to capture syntactic properties, e.g., the arc label, the direction of the head and dependents, and the relative order of dependents.

Finally, supertagging was presented as a method to provide syntactic information to the feature model of a statistical dependency parser (Ambati et al., 2014; Ouchi et al., 2014). Ambati et al. (2014) used for this reason CCG categories, and Ouchi et al. (2014) extracted the supertag tag set from a dependency treebank. In both cases, the models that used the additional features achieved significant improvements over their baseline.

In this dissertation, we adopt the interpretation of supertagging proposed by Ambati et al. (2014) and Ouchi et al. (2014) and take supertagging as a way of incorporating

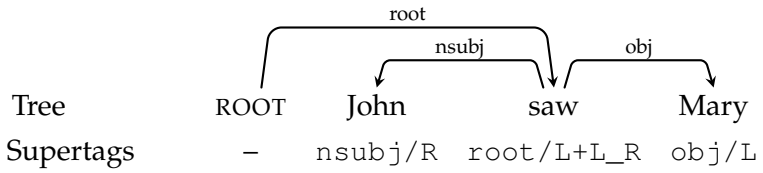


Figure 4.1: Supertags derived from a dependency tree. They encode the label, the head direction, and the presence of left and right dependents.

syntactic features instead of the traditional use for disambiguation. To explain the concept further, Figure 4.1 gives an example of a dependency tree for the sentence "*John saw Mary*" and possible supertags that can be extracted from it. These supertags follow the Model 1 of Ouchi et al. (2014) and encode information about the label, the head direction, and the presence of left and right dependents. For example, the word *saw* receives the supertag `root/L+L_R` because its head (label `root`) is on the left (`root/L`), and the word has two modifiers, one on each side (`L_R`).

Supertagging vs. stacking. Now that we explained supertagging, we look into its relation to feature-based stacking. Both supertagging and stacking are known to be effective, and despite the difference in conceptual motivation, the pieces of information they contribute to parsing decisions are intuitively very similar. When supertags were first proposed by Joshi and Bangalore (1994), they called supertagging *almost parsing*, because supertags anticipate much syntactic disambiguation. In stacking, the first step is running a parser, or in other words, *real parsing*. This chapter investigates the difference between *almost* and *real parsing* for improving a statistical dependency parser.

We compare the two methods – supertagging and stacking – on ten different datasets. We discuss results on a large number of comparative experiments with two well-established dependency parsers (a graph-based and a transition-based) and a sequence labeler. In the first set of experiments, we use only the two parsers and compare supertagging and stacking in artificial and realistic settings (Section 4.2). In the second set of experiments, we control for the methods and compare different ways of realizing them (Section 4.3). In the last set, we evaluate the benefit of combining both methods (Section 4.4).

Intuitively, stacking should give more significant improvements than supertagging, because trees in stacking are more informative than supertag sequences in supertagging. However, our experiments show that both methods perform on par. Based on an in-depth analysis of these findings, we argue that supertagging is a form of feature-based

stacking.

One apparent advantage of supertagging is the fact that one can predict supertags without a parser and thus possibly faster. However, greedy transition-based parsers are extremely fast as well. We show that the outputs of a sequence labeler and a greedy transition-based parser are of equal usefulness when used in supertagging. This setup has a clear potential for application to large-scale (e.g., web) data. Indeed, we test supertagging and stacking on the English Web Treebank (Bies et al., 2012) and show that both methods also improve parsing in a cross-domain setting.

Our experiments on combining supertagging and stacking result in small gains only when different tools predict supertags and trees. This is in line with the finding of Surdeanu and Manning (2010), who demonstrated that the diversity of algorithms is essential when stacking parsers. Since supertagging is a form of stacking, this also holds for supertagging, and we argue that this is a more important factor than the choice between the two methods.

4.1 Experimental Setup

This section describes the experimental setup used for evaluation. We start by describing treebanks and preprocessing steps. We then give details of the two parsers used in the experiments. Specifically, we explain how these parsers' feature models were extended to incorporate information coming from different Level 0 models.

4.1.1 Datasets and Preprocessing

Datasets. We perform experiments on the treebanks from the SPMRL 2014 shared task on parsing morphologically-rich languages (Seddah et al., 2014). The treebanks cover nine languages: Arabic, Basque, French, Hebrew, German, Hungarian, Korean, Polish, and Swedish (see Appendix A.1 for references to particular treebanks and their statistics). Since all these languages are morphologically-rich, we extend this collection by adding English as an example of a language with minimal inflection. Specifically, we use the English Penn Treebank (Marcus et al., 1993) converted to Stanford Dependencies.[2] Sections 2-21 are used for training, section 24 as a development set, and section 23 as a test set.

[2]For the conversion, we use the Stanford Parser version 3.4.1 `http://nlp.stanford.edu/software/lex-parser.shtml`.

Preprocessing. We follow Björkelund et al. (2013), i.e., the winning team of the above-mentioned shared task, and apply several language-specific preprocessing steps to ensure high parsing performance across all languages. Specifically, fine-grained POS tags are used for Arabic, French, Hebrew, and Swedish, and coarse-grained for all the other languages. Moreover, we reduce the Hungarian dependency label set to the 50 most frequent labels and map the composite labels to their base label.[3]

Throughout the whole dissertation, we apply the standard NLP pipeline that we described in Section 2.2.1. We assume gold-standard sentence segmentation and tokenization in all experiments. Other preprocessing information, such as POS tags or morphological features, is predicted automatically.

Traditionally, sequence labeling tasks have been addressed with probabilistic models, such as Hidden Markov Models (HMM) and Conditional Random Fields (CRF). We use Mueller et al.'s (2013) CRF-based tagger MARMOT. This tagger predicts POS and morphological tags jointly. While more recent taggers are based on neural mechanisms such as BiLSTMs (Plank et al., 2016), MARMOT still achieves comparable performance to the neural tools on POS and morphological tagging tasks. For example, in the CoNLL 2017 UD shared task (Zeman et al., 2017), our submission used MARMOT and achieved the second place in the task of POS tagging and the first place in predicting morphological information (Björkelund et al., 2017).

We annotate the training sets via 5-fold jackknifing with POS tags, morphological features, and lemmas. Lemmatization is performed with mate-tools (Bohnet, 2010; Bohnet and Nivre, 2012).[4] Moreover, to improve POS tagging accuracy, we integrate the output of language-specific morphological analyzers as additional features into MARMOT (see Table 4.1).

4.1.2 Supertag Design

Deciding which supertag model to use amounts to deciding how much information to encode in each supertag. Foth et al. (2006) showed that richer supertags have a bigger positive influence on parsing accuracy. The level of granularity of a model, however, increases with the difficulty of automatically predicting supertags.

Ouchi et al. (2014) tested two models. Model 1 (see Figure 4.1) exploited the relative position of the head of a word (`hdir`), its dependency relation (`label`), and information

[3] The Hungarian dataset encodes ellipses through composite labels. As a result, it contains more than 400 labels (as compared to the average 32 for other SPMRL treebanks), which are sparse and difficult to learn for a parser.

[4] `https://code.google.com/archive/p/mate-tools/`

Language	Morphological analyzer
Arabic	AraMorph, a re-implementation of Buckwalter (2002)
Basque	Apertium (Forcada et al., 2011)
French	An extension of Zhou (2007)
Hebrew	Analyzer from Goldberg and Elhadad (2013)
German	SMOR (Schmid et al., 2004)
Hungarian	Magyarlanc (Zsibrita et al., 2013)
Korean	HanNanum (Park et al., 2010)
Polish	Morfeusz (Woliński, 2006)
Swedish	Granska (Domeij et al., 2000)

Table 4.1: Morphological analyzers used for preprocessing.

whether the word's dependents occur to the right or to the left of the word (`hasLdep`, `hasRdep`). Model 2 additionally used dependency relations of obligatory dependents of verbs. The difference between the accuracy of parsers using these two models on the test set was not significant. Moreover, in (Björkelund et al., 2014), we performed experiments with the same models but other languages and parsers. The result was similar - the richer models usually resulted in better accuracy, but the differences were negligible.

Based on these results, we decided to use Model 1 in all of the experiments. The supertags are extracted from the respective training sets and follow the template `label/hdir+hasLdep_hasRdep`.

4.1.3 Notation and Evaluation

We denote stacking and supertagging by STACK and STAG, respectively. When a tool Y uses the output of another tool X, we mark this by superscript and subscript. For example, $\text{STACK}^{\text{Y}}_{\text{X}}$ means that tool Y uses the output of tool X in stacking. Similarly, $\text{STAG}^{\text{Y}}_{\text{X}}$ means that tool Y uses the supertags predicted by tool X. Moreover, we follow the terminology from classifier stacking and call X the Level 0 tool and Y the Level 1 tool.

We evaluate the parsing experiments using Labeled Attachment Score (LAS), a standard in dependency parsing metric. It calculates the ratio of tokens with a correct head and label to the total number of tokens in the test data. We mark statistical significance against respective baselines by ‡ and †, denoting p-value < 0.05 and p-value < 0.01, respectively. Significance testing is carried out using the Wilcoxon signed-rank test. Averages and oracle experiments are not tested for significance.

4.1.4 Parsers and Feature Models

Since we are interested in the effect of supertagging and stacking on the parsing performance, in all experiments Level 1 tools are always dependency parsers. We experiment with one graph-based and one transition-based parser to cover the two dominant paradigms in dependency parsing.

We extend the parsers' baseline feature sets in two directions: (1) by extracting features for stacking, i.e., features from a provided dependency tree, and (2) by extracting features from a sequence of supertags. For stacking, the features are taken from Nivre and McDonald (2008) and slightly adapted to our experimental setting. For supertagging, we mirror the features from stacking to the best extent possible, given the more limited information that is contained in the supertags.

We note that feature engineering can be carried out more elaborately for both stacking (Martins et al., 2008) and supertagging (Ouchi et al., 2014). However, since not all types of features necessarily carry over from one method to the other, a simpler feature set is more useful for comparison. Moreover, both Martins et al. (2008) and Ouchi et al. (2014) achieved only minor performance gains with more elaborate features. Below, we describe our transition- and graph-based parsers and the corresponding feature set extensions.

Transition-based parser. As our transition-based parser, we use the IMSTRANS parser from Björkelund and Nivre (2015). The parser uses the ARCSTANDARD system extended with a SWAP transition to handle non-projective structures (Nivre, 2009), the lazy SWAP oracle by Nivre et al. (2009), and beam search with beam size of 20. It is trained with a globally optimized structured perceptron (Zhang and Clark, 2008) and the passive-aggressive update (Crammer et al., 2006). The baseline feature set of this parser is primarily based on Zhang and Nivre's (2011) feature model (see Table 2.2) with adaptations to the ARCSTANDARD setting.

Table 4.2 outlines the feature templates used for stacking and supertagging. The predicates $\text{head}_X(d)$, $\text{hdir}_X(d)$, $\text{label}_X(d)$, $\text{hasL}_X/\text{hasR}_X(d)$, and $\text{stag}_X(d)$ extract the head of d, the direction of d's head, the arc label of d, whether d has left/right dependents, and the supertag according to the Level 0 prediction X. X is either a dependency graph (in stacking) or a supertag assignment (in supertagging), denoted G and S in Table 4.2, respectively. Subscripts $._L$ and $._R$ mark the leftmost/rightmost dependents of tokens given the current parser state. $\text{POS}(d)$ extracts the POS tag of d, with a special placeholder if d is undefined.

The stacking features are mostly based on the features from Nivre and McDonald

Stacking features	
$\text{head}_\text{G}(s_0) = s_1$	$\text{head}_\text{G}(s_1) = s_0$
$\text{hdir}_\text{G}(s_0)$	$\text{hdir}_\text{G}(s_1)$
$\text{label}_\text{G}(s_0)$	$\text{label}_\text{G}(s_1)$
$\text{hasL}_\text{G}(s_0) \circ \text{POS}(s_{0L})$	$\text{hasR}_\text{G}(s_0) \circ \text{POS}(s_{0R})$
$\text{hasL}_\text{G}(s_1) \circ \text{POS}(s_{1L})$	$\text{hasR}_\text{G}(s_1) \circ \text{POS}(s_{1R})$
Supertag features	
$\text{stag}_\text{S}(s_0)$	$\text{stag}_\text{S}(s_1)$
$\text{label}_\text{S}(s_0)$	$\text{label}_\text{S}(s_1)$
$\text{hdir}_\text{S}(s_0)$	$\text{hdir}_\text{S}(s_1)$
$\text{hasL}_\text{S}(s_0)$	$\text{hasR}_\text{S}(s_1)$
$\text{stag}_\text{S}(s_0) \circ \text{POS}(s_{0L})$	$\text{stag}_\text{S}(s_0) \circ \text{POS}(s_{0R})$
$\text{stag}_\text{S}(s_1) \circ \text{POS}(s_{1L})$	$\text{stag}_\text{S}(s_1) \circ \text{POS}(s_{1R})$
$\text{hasL}_\text{S}(s_0) \circ \text{hdir}_\text{S}(s_1)$	$\text{hasR}_\text{S}(s_1) \circ \text{hdir}_\text{S}(s_0)$
$\text{hasL}_\text{S}(s_0) \circ \text{POS}(s_{0L})$	$\text{hasR}_\text{S}(s_1) \circ \text{POS}(s_{1R})$
$\text{hasR}_\text{S}(s_0) \circ \text{POS}(s_{0R})$	$\text{hasL}_\text{S}(s_1) \circ \text{POS}(s_{1L})$

Table 4.2: Feature templates used for stacking and supertagging in the transition-based parser. $\circ$ denotes conjunctions of basic templates. All templates are conjoined with the POS tag of the topmost stack items s_0 and s_1.

(2008), with the exception of the last two rows (cf., Table 3.1 in Chapter 3). The additional features encode whether d should have left/right dependents according to G conjoined with information whether d has left/right dependents in the current configuration. We added them because the existence of left/right dependents is also encoded in the supertags (see Section 4.1.2). Conjoining the existence of left/right dependents according to the Level 0 predictions with the POS tag of left/right dependents in the current parser state thus encodes whether dependents were attached or not. Since the ARCSTANDARD algorithm works bottom-up, every token needs to collect all its dependents before it can be attached to its own head.

The supertag features mimic the information provided by stacking. For instance, in stacking, the Level 0 predictions explicitly include whether s_0 is the head of s_1. In supertagging, this is approximated by combining the direction of the head of s_1 with whether s_0 expects dependents on the left.

Stacking features	
$\text{head}_G(d) = h$	
$\text{label}_G(d)$	
$\text{head}_G(d) = h \circ \text{label}_G(d)$	
Supertag features	
$\text{stag}_S(h)$	$\text{stag}_S(d)$
$\text{label}_S(d)$	$\text{hdir}_S(d)$
$\text{label}_S(d) \circ \text{hdir}_S(d)$	
$\text{hasL}_S(h)$	$\text{hasR}_S(h)$
$\text{hdir}_S(d) \circ \text{hasL}_S(h)$	$\text{hdir}_S(d) \circ \text{hasR}_S(h)$
$\text{label}_S(d) \circ \text{hasL}_S(h)$	$\text{label}_S(d) \circ \text{hasR}_S(h)$
$\text{label}_S(d) \circ \text{hdir}_S(d) \circ \text{hasL}_S(h)$	
$\text{label}_S(d) \circ \text{hdir}_S(d) \circ \text{hasR}_S(h)$	

Table 4.3: Feature templates used for stacking and supertagging in the graph-based parser. $\circ$ denotes conjunctions of basic templates. All templates are conjoined with the direction of that arc and with the POS tag of the head and the dependent.

Graph-based parser. As a graph-based parser we use the TurboParser.[5] This parser solves the parsing task by performing global inference with a dual decomposition algorithm. It outputs non-projective structures natively (Martins et al., 2013). We train it with third-order features.[6]

Table 4.3 shows the stacking and supertagging features as we implemented them in TurboParser. Where possible, they are synchronized with the features for the transition-based parser. We extract these features only on first-order factors, with d and h denoting the dependent and the head, respectively. Unlike in (Nivre and McDonald, 2008), the features cannot access the label of the current arc during feature extraction, because it is automatically combined with the features only after the extraction.

Like in the transition-based parser, supertag and stacking features are modeled to capture similar information. However, features that combine information about dependents of dependents with information about the head are not included since these would require higher-order factors.

[5] We use TurboParser version 2.0.1 from `http://www.ark.cs.cmu.edu/TurboParser/`

[6] We set MODELTYPE=FULL while training the TurboParser models.

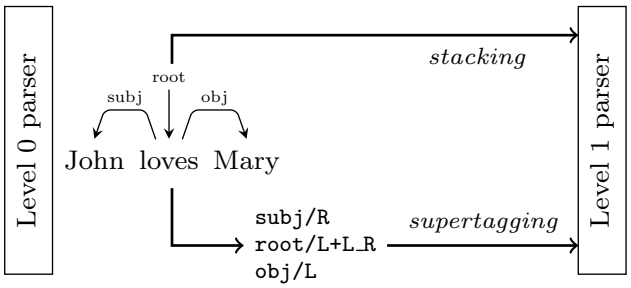


Figure 4.2: Setup for comparing stacking and supertagging.

4.2 Comparing Supertagging and Stacking

We start our experiments by comparing the performance of supertagging and stacking with the aim to derive some conclusions about their relationship to each other. The experimental setup is illustrated in Figure 4.2. We use one parser as Level 0 and the other as Level 1 tool. In stacking, the Level 1 parser exploits the tree produced by the Level 0 parser as additional features. In supertagging, we derive the supertag of each token from the tree that is output by the Level 0 parser. The Level 1 parser then uses these supertags as additional features. Although supertags are typically predicted with sequence labelers, using a parser on Level 0 in both cases ensures that the only difference between the two settings is how the information is given to the Level 1 parser, i.e., as a tree or as a sequence of supertags (we experiment with sequence labelers as supertaggers in Section 4.3).

The training sets are annotated with predicted dependency trees or supertags via 5-fold jackknifing. In the tables, **GB** stands for the graph-based parser and **TB** for the transition-based parser.

4.2.1 Supertagging and Stacking Accuracy

Table 4.4 gives the performance of the Level 1 parser on the test sets. In the baseline setting (**BL**), the parser is run without any additional information. STAG and STACK show the performance of the Level 1 parser when provided with supertags or a tree from the Level 0 parser.

As demonstrated by previous work, both stacking and supertagging improve the parsing performance of the Level 1 parser consistently. For supertagging, our results confirm the findings by Ouchi et al. (2014) and Ambati et al. (2014). The stacking results are in line with Nivre and McDonald (2008) and Martins et al. (2008). Interestingly, both

	Graph-based			Transition-based		
	BL^{GB}	$\text{STAG}^{\text{GB}}_{\text{TB}}$	$\text{STACK}^{\text{GB}}_{\text{TB}}$	BL^{TB}	$\text{STAG}^{\text{TB}}_{\text{GB}}$	$\text{STACK}^{\text{TB}}_{\text{GB}}$
Arabic	84.99	85.54†	85.65†	85.09	85.58†	85.60†
Basque	82.79	83.24†	83.32†	81.77	82.99†	83.14†
English	90.12	90.99†	90.95†	90.54	90.98†	90.88†
French	83.95	84.33†	84.29†	83.47	83.88†	84.16†
German	88.53	89.14†	89.15†	87.89	88.88†	88.91†
Hebrew	79.41	80.10‡	80.03†	79.70	80.04	80.30
Hungarian	83.98	85.52†	85.46†	85.25	85.37	85.38
Korean	86.18	86.48	86.59†	85.71	86.31†	86.06‡
Polish	84.96	85.48	85.52‡	84.34	85.03	85.06‡
Swedish	79.59	80.63†	80.66†	79.97	81.04†	81.32†
average	84.45	85.15	85.16	84.37	85.01	85.08

Table 4.4: Parsing results (LAS) on test sets from the SPMRL 2014 shared task.

methods improve the parsing accuracy to the same extent, with the average improvements about 0.7 LAS for both the graph-based and the transition-based parser. Almost all of the improvements are statistically significant, with a few exceptions, most notably Polish.

So far, we have established that both stacking and supertagging improve parsers to the same extent. We now proceed with more in-depth analysis aimed at finding where the improvements are coming from. We perform the analysis on the development sets in order to not compromise our test sets. The development set counterpart to Table 4.4 is displayed in the first three columns of Tables 4.5 and 4.6, for graph-based and transition-based Level 1 parsers, respectively. The test and development results show the same patterns when it comes to the supertagging and stacking results.

4.2.2 In-Depth Analysis

The previous experiments show that the impacts of supertagging and stacking on parsing accuracy go in parallel. However, although the overall results of these two approaches are similar, they might still come about in different ways. To investigate this, we follow McDonald and Nivre (2007) and look into the accuracy distributions relative to sentence length and dependency length, i.e., the distance between the dependent and the head. We present the analysis on the concatenation of all the development sets. We also looked

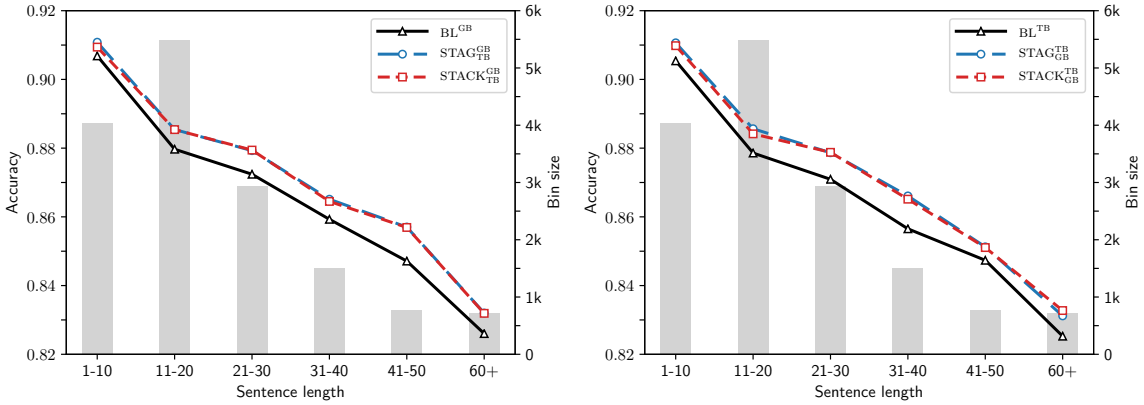


Figure 4.3: The accuracy of the graph-based (left) and transition-based (right) parser relative to sentence length.

at the corresponding plots for the individual treebanks. While the absolute numbers vary across the different datasets, the relative differences between the baseline, supertagging, and stacking models are consistent with the concatenation.

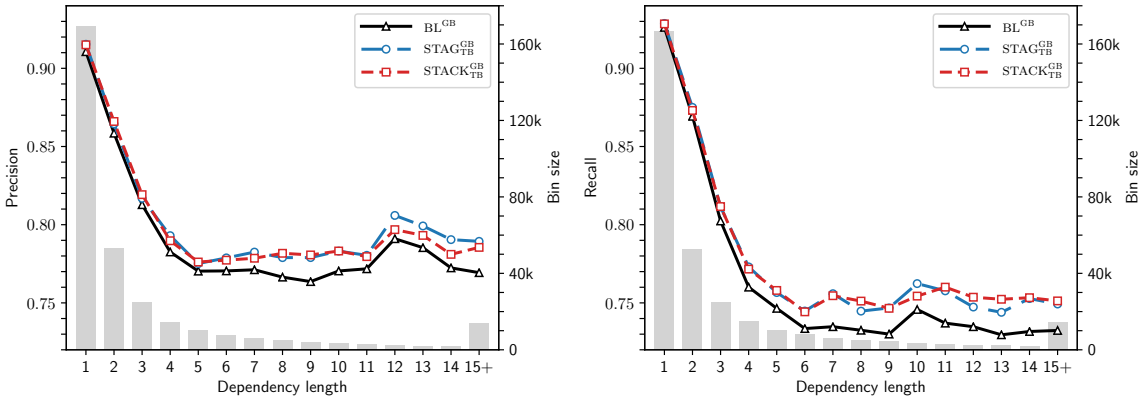
Figure 4.3 reports the accuracy of both parsers relative to sentence length in bins of size 10. Bin sizes are represented as grey bars.[7] Both charts show that stacking and supertagging improve accuracy in a similar fashion across all bins.

Figure 4.4 displays both parsers' performance relative to the dependency length in terms of precision and recall. Precision is defined as the percentage of correct predictions among all predicted arcs of length l and recall is the percentage of correct predictions among gold standard arcs of length l.[8] In all plots, the stacked and supertagged systems show consistent improvement over the baseline. Moreover, the curves of the stacked and supertagged systems are mostly parallel and close to each other.

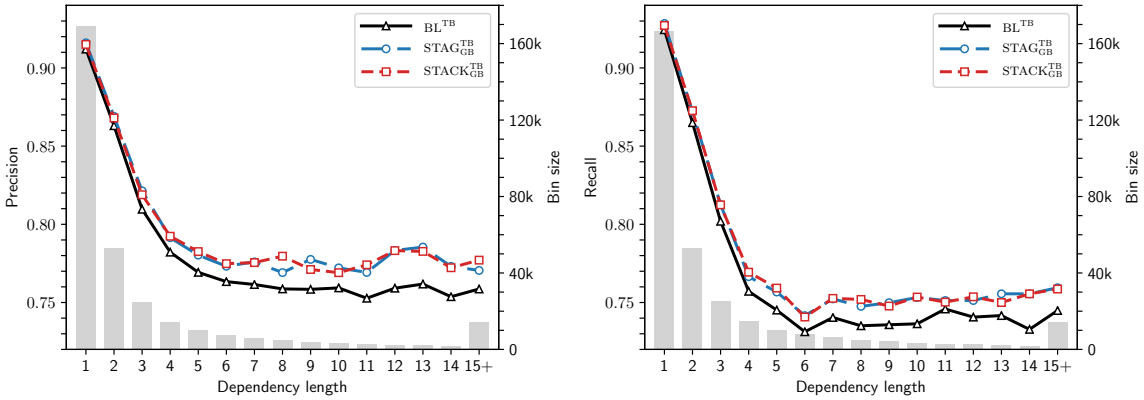
Supertagging and stacking do not just appear similar at the macro level in terms of LAS. The analysis shows that their contributions are also very similar when broken down by dependency or sentence length. Moreover, the improvements are not restricted to sentences or arcs of particular lengths. Therefore, we conclude that both methods are indeed doing the same thing. We now move on to additional experiments that are meant to investigate the similarity between supertagging and stacking further.

[7] Note that if there are fewer items in a bin, the curves are more sensitive to small absolute changes.

[8] For precision, the bin sizes shown as grey bars are averages over all three systems, as the number of predicted arcs of a certain length can vary.



(a) Graph-based parser



(b) Transition-based parser

Figure 4.4: The dependency precision (left) and recall (right) relative to dependency length.

4.2.3 Oracle Experiments

In order to assess the potential utility of supertags, we provided the parsers with gold supertags. We expect the gold supertags to give a considerable boost to accuracy as they encode correct syntactic information. Intuitively, one would expect the corresponding stacking experiment (providing gold trees) to reach 100% accuracy since the parser receives the full solution as features. However, this assumption turns out not to hold.

Columns STAG_O in Tables 4.5 and 4.6 show the results for the supertag experiments. Comparing the results with the first BL column, we find big jumps (between 7 and 20 LAS) in performance. For German, English, and Polish performance goes up even to 97/98 LAS. These huge jumps are due to the amount of syntactic information encoded in the supertags, which is much higher than in POS tags, for example.

Columns STACK_O in Tables 4.5 and 4.6 show the results for the stacking experiments.

		Other parser		Oracle		Self-application	
	BL^{GB}	$STAG^{GB}_{TB}$	$STACK^{GB}_{TB}$	$STAG^{GB}_{O}$	$STACK^{GB}_{O}$	$STAG^{GB}_{GB}$	$STACK^{GB}_{GB}$
Arabic	85.64	86.25†	86.25†	93.78	98.67	85.83†	85.69‡
Basque	83.37	83.96†	83.78†	96.66	96.87	83.68†	83.44
English	88.59	89.47†	89.41†	97.91	98.34	88.77†	88.62
French	84.78	85.05‡	85.14†	96.43	98.38	84.91	84.80
German	91.46	92.10†	92.01†	98.60	98.90	91.64†	91.49
Hebrew	78.71	79.36‡	79.56†	94.43	92.62	79.31†	78.90‡
Hungarian	82.60	83.92†	83.72†	94.37	94.21	82.87†	82.64
Korean	86.08	86.24	86.34‡	93.48	96.46	86.28†	86.08
Polish	85.17	85.52‡	85.29	97.41	96.55	85.18	85.24
Swedish	75.24	76.27‡	76.44†	94.22	93.33	75.89†	75.39
average	84.16	84.81	84.79	95.73	96.43	84.44	84.23

Table 4.5: Results (LAS) on development sets for stacking and supertagging with different Level 0 information sources and Level 1 graph-based parser.

		Other parser		Oracle		Self-application	
	BL^{TB}	$STAG^{TB}_{GB}$	$STACK^{TB}_{GB}$	$STAG^{TB}_{O}$	$STACK^{TB}_{O}$	$STAG^{TB}_{TB}$	$STACK^{TB}_{TB}$
Arabic	85.69	86.06†	86.19†	94.09	98.80	85.77	85.73
Basque	82.22	84.02†	83.86†	97.65	97.08	82.49	82.24
English	89.06	89.48†	89.30	98.16	98.42	89.01	89.22
French	84.07	84.68†	84.55†	96.64	98.62	84.05	84.29‡
German	91.15	91.98†	91.98†	98.80	98.65	91.44†	91.35†
Hebrew	78.80	79.82	79.76	95.70	92.96	78.69	78.67
Hungarian	83.27	83.83†	83.87†	96.46	96.14	83.61†	83.45‡
Korean	85.97	86.56†	86.25	94.90	97.27	85.92	85.86
Polish	84.51	85.73‡	85.65	98.50	97.21	84.84	84.76
Swedish	75.65	77.20†	77.16†	95.65	93.18	75.78	75.66
average	84.04	84.94	84.86	96.66	96.83	84.16	84.12

Table 4.6: Results (LAS) on development sets for stacking and supertagging with different sources of Level 0 information and Level 1 transition-based parser.

Surprisingly, stacking with gold dependency trees does not reach 100% accuracy. Moreover, comparing columns STAG_{O} and STACK_{O}, we find that, on average, supertagging and stacking improve parser's performance to the same extent.

The fact that gold supertags do not yield maximum accuracy is not so surprising. A supertag sequence does not encode the full dependency tree but merely indicates the direction of heads and dependents. However, it is puzzling that stacking with gold trees does not lead to perfect parsing results. In the case of the transition-based parser, the reason might be that the parser does not carry out an exhaustive search but uses a beam search to explore only a fraction of the search space. That is, the gold solution can get pruned early enough that the parser never considers it. For the graph-based parser, this result is more unexpected since this parser does exact search. We hypothesize that the lack of regularization during training might assign enough weight to the regular features such that they can override the few stacking features that convey the correct solution.

4.2.4 Self-Application

Two last columns in Tables 4.5 and 4.6 report the results of experiments in which we use the same parser at Level 0 and Level 1. We know from Martins et al. (2008) that self-application, i.e., stacking a parser on its own output, leads to at most tiny improvements, especially compared to settings in which different parsers are stacked. Our results corroborate these findings. More interestingly, we find a similar effect for supertagging.[9] This effect demonstrates that it is crucial that Level 0 and Level 1 use different ways of modeling the data to benefit from the combination (cf., Surdeanu and Manning (2010)). Whether the combination is done via supertagging or stacking is of less importance, since both methods provide similar improvements.

4.3 Supertagging without Parsers

One potential advantage of supertagging over stacking is the fact that one can predict supertags without a parser. Most previous work predicts supertags using classifiers or sequence models, which is the standard approach for tagging problems. As tagging is commonly considered an "easier" task than parsing, one could assume that supertags can be predicted very efficiently using standard sequence labeling algorithms. But sequence labelers would not be able to predict the dependency tree in a stacking setup.

[9] Note that most of the improvements in $\text{STAG}^{\text{GB}}_{\text{GB}}$ are actually statistically significant. However, the difference to BL^{GB} is considerably smaller than in the setting $\text{STAG}^{\text{GB}}_{\text{TB}}$ with different parsers (avg. difference is 0.28 LAS vs. 0.65 LAS points absolute).

	GB	TB	SL	GTB
Arabic	90.44	90.81	87.90	89.08
Basque	83.71	82.80	79.50	80.91
English	91.03	91.46	86.38	89.47
French	87.37	86.95	82.75	85.00
German	90.97	90.92	85.39	88.27
Hebrew	80.81	81.52	76.59	79.61
Hungarian	83.65	84.71	79.55	81.91
Korean	90.37	90.06	90.77	89.66
Polish	85.00	84.93	81.46	81.78
Swedish	75.16	76.53	72.15	74.53
average	85.85	86.07	82.24	84.02

Table 4.7: Supertag tagging accuracy on development sets.

The two parsers that we use in the experiments are indeed unlikely to outperform standard sequence labelers in terms of speed. However, greedy ARCSTANDARD parsers are very fast. Therefore, in the next experiment, we compare a greedy ARCSTANDARD parser, i.e., the IMSTRANS parser without beam search, with MARMOT. We follow Ouchi et al. (2014) in adding POS tags and morphological information to the feature model of the sequence labeler.

This experiment's purpose is two-fold: So far, we predicted supertags by predicting a tree first and then deriving the supertags from that tree. Now, we test how our previous results compare to supertags predicted by a sequence labeler, which is the conventional way of predicting supertags. Furthermore, we want to see how supertagging with a sequence labeler compares to supertagging and stacking with a parser that is equally efficient.

4.3.1 Comparison with Level 0 Parsers

We start by looking at *tagging* accuracy of the four tools used as supertag predictors, i.e., two regular parsers, sequence labeler (denoted **SL**), and greedy transition-based parser (denoted **GTB**). Results provided in Table 4.7 show that the graph- and transition-based parsers are roughly equal on average, whereas the sequence labeler is clearly behind by 3.6% points absolute. The greedy parser falls in-between with an accuracy of 84.02. Next, we analyze how these big differences in tagging accuracy impact the parsing accuracy of Level 1 parsers.

		Labeler	Greedy parser	
	BL^{GB}	$\text{STAG}^{\text{GB}}_{\text{SL}}$	$\text{STAG}^{\text{GB}}_{\text{GTB}}$	$\text{STACK}^{\text{GB}}_{\text{GTB}}$
Arabic	85.64	86.24†	86.16†	86.24†
Basque	83.37	84.33†	83.56	83.57
English	88.59	89.06†	89.12†	89.09†
French	84.78	84.89	85.02‡	85.11‡
German	91.46	91.89†	91.97†	91.97†
Hebrew	78.71	79.82†	79.41†	79.50†
Hungarian	82.60	83.54†	83.36†	83.22†
Korean	86.08	86.85†	86.07	86.22
Polish	85.17	85.89‡	85.29	85.45
Swedish	75.24	76.62†	76.56†	76.26‡
average	84.16	84.91	84.65	84.66

(a) Graph-based parser

		Labeler	Greedy parser	
	BL^{TB}	$\text{STAG}^{\text{TB}}_{\text{SL}}$	$\text{STAG}^{\text{TB}}_{\text{GTB}}$	$\text{STACK}^{\text{TB}}_{\text{GTB}}$
Arabic	85.69	85.81	85.70	85.84‡
Basque	82.22	83.58†	82.44	82.46
English	89.06	88.91	88.81	88.95
French	84.07	84.07	84.16	84.19
German	91.15	91.37‡	91.31	91.22
Hebrew	78.80	79.86†	79.38	78.93
Hungarian	83.27	83.91†	83.16	83.30
Korean	85.97	86.98†	85.91	85.77
Polish	84.51	84.93	84.59	84.92
Swedish	75.65	76.87‡	76.12	76.12
average	84.04	84.63	84.16	84.17

(b) Transition-based parser

Table 4.8: Results (LAS) on development sets with a sequence labeler and a greedy transition-based parser as Level 0 tools.

Sequence labeler. Columns STAG_{SL} in Table 4.8 show that, on average, parsing performance is not harmed by predicting supertags with the sequence labeler instead of one of the parsers (compare to columns $\text{STAG}^{\text{GB}}_{\text{TB}}$ and $\text{STAG}^{\text{TB}}_{\text{GB}}$ in Tables 4.5 and 4.6). It depends on the individual dataset, whether the sequence labeler is more useful than one of the parsers. However, in general, the supertags predicted by the sequence labeler improve parsing performance to a similar extent.

Greedy parser. The experiments with the greedy parser yield different results for the graph-based and the transition-based parser on Level 1: When the graph-based parser acts as Level 1, the greedy parser is slightly behind the sequence labeler in terms of performance. This holds both for supertagging and stacking experiments (compare the column $\text{STAG}^{\text{GB}}_{\text{SL}}$ to columns $\text{STAG}^{\text{GB}}_{\text{GTB}}$ and $\text{STACK}^{\text{GB}}_{\text{GTB}}$), which again suggests that supertagging and stacking are interchangeable. However, when Level 1 is the transition-based parser, we find a self-application effect for the greedy parser, both in supertagging and stacking (columns BL^{TB} vs. $\text{STAG}^{\text{TB}}_{\text{GTB}}$ and $\text{STACK}^{\text{TB}}_{\text{GTB}}$). This is not surprising since the decoding algorithms in the beam-search and greedy transition-based parser are identical; it simply underlines the importance of having different algorithms in the setup.

4.3.2 Out-of-Domain Application

The previous experiment shows that the greedy parser at Level 0 gives competitive results compared to a sequence labeler. Having fast predictors available for stacking or supertagging suggests an application where speed matters, e.g., Ambati et al. (2014) propose supertags to improve the performance of fast parsers in a web-scale scenario.

As web data contain very heterogeneous types of text, the question is whether the positive effects of supertagging and stacking are actually preserved in such an out-of-domain setting. To test this, we conduct experiments on the English Web Treebank (Bies et al., 2012) converted to Stanford Dependency format. All the models are trained on sections 2-21 from the English Penn Treebank.

The results in Tables 4.9a and 4.9b show consistent improvements in the five genres of the dataset both for supertagging and stacking. Both are thus suitable methods to improve parsing accuracy when parsing out-of-domain data. Since parsing speed also depends on the Level 1 parser, a greedy transition-based parser would be preferable in such an application. Using supertagging with a sequence labeler to provide syntactic information to the greedy parser is then the right choice because it avoids a self-application effect.

		Labeler	Parser	Greedy parser	
	BL^{GB}	$\text{STAG}^{\text{GB}}_{\text{SL}}$	$\text{STACK}^{\text{GB}}_{\text{TB}}$	$\text{STAG}^{\text{GB}}_{\text{GTB}}$	$\text{STACK}^{\text{GB}}_{\text{GTB}}$
question-answers	74.09	74.52†	74.88†	74.57‡	74.55†
email	75.06	75.75†	75.72†	74.91	75.00‡
newsgroups	76.16	76.88†	76.49	76.68†	76.71†
reviews	76.32	76.99†	77.10†	76.95†	77.01†
weblogs	79.78	79.98	80.44†	80.35†	80.39†
average	76.28	76.82	76.93	76.69	76.73

(a) Graph-based parser

		Labeler	Parser	Greedy parser	
	BL^{TB}	$\text{STAG}^{\text{TB}}_{\text{SL}}$	$\text{STACK}^{\text{TB}}_{\text{GB}}$	$\text{STAG}^{\text{TB}}_{\text{GTB}}$	$\text{STACK}^{\text{TB}}_{\text{GTB}}$
question-answers	74.41	74.37	74.64	74.13	74.13
email	75.16	75.85†	75.68†	74.44	74.62
newsgroups	76.09	76.61‡	76.96†	76.18	76.09
reviews	76.76	77.06	77.14†	76.47	76.72‡
weblogs	80.13	80.28	81.05†	79.84	79.75
average	76.51	76.83	76.83	76.21	76.26

(b) Transition-based parser

	BL^{GTB}	$\text{STAG}^{\text{GTB}}_{\text{SL}}$
question-answers	72.32	72.75
email	73.25	73.86†
newsgroups	74.00	74.88†
reviews	74.73	75.33†
weblogs	77.79	78.24†
average	74.42	75.01

(c) Greedy transition-based parser

Table 4.9: Stacking and supertagging results (LAS) on the English Web Treebank for three different Level 1 parsers.

	Graph-based parser			Transition-based parser		
	BL^{GB}	MAX^{GB}	$\text{COMBINE}^{\text{GB}}$	BL^{TB}	MAX^{TB}	$\text{COMBINE}^{\text{TB}}$
Arabic	85.64	86.25†	86.62†	85.69	86.19†	86.43†
Basque	83.37	84.33†	84.51†	82.22	83.86†	84.09†
English	88.59	89.41†	89.74†	89.06	89.30	89.67†
French	84.78	85.14†	85.35†	84.07	84.55†	84.59†
German	91.46	92.01†	92.22†	91.15	91.98†	92.04†
Hebrew	78.71	79.82†	79.90†	78.80	79.86†	79.75‡
Hungarian	82.60	83.72†	84.23†	83.27	83.91†	84.08†
Korean	86.08	86.85†	86.67†	85.97	86.98†	87.32†
Polish	85.17	85.89‡	85.78‡	84.51	85.65	86.01‡
Swedish	75.24	76.62†	76.98†	75.65	77.16†	77.30†
average	84.16	85.00	85.20	84.04	84.94	85.13

Table 4.10: Results (LAS) on development sets for combining supertags and stacking. MAX^{GB} corresponds to $\max(\text{STAG}^{\text{GB}}_{\text{SL}}, \text{STACK}^{\text{GB}}_{\text{TB}})$, MAX^{TB} to $\max(\text{STAG}^{\text{TB}}_{\text{SL}}, \text{STACK}^{\text{TB}}_{\text{GB}})$, $\text{COMBINE}^{\text{GB}}$ to $(\text{STAG}_{\text{SL}}{+}\text{STACK}_{\text{TB}})^{\text{GB}}$, and $\text{COMBINE}^{\text{TB}}$ to $(\text{STAG}_{\text{SL}}{+}\text{STACK}_{\text{GB}})^{\text{TB}}$.

Table 4.9c shows the performance when the greedy parser is acting as Level 1. Supertagging improves over the baseline significantly on 4 out of 5 datasets. However, the baseline for the greedy parser is, on average, about 2 LAS behind the other two parsers. This loss in accuracy buys a significant speed-up, though. The greedy parser is about 29 times faster[10] than the graph-based parser on the English dataset and even 80 times faster on the Arabic dataset. As the Arabic dataset has very long sentences, the higher complexity of the graph-based parser has a notable effect on its performance. Compared to the beam-search transition-based parser, the greedy parser is about ten times faster on English and five times faster on Arabic.

4.4 Combining Supertagging and Stacking

Previous work and our experiments show that supertagging and stacking both improve the accuracy of a parser. It is therefore a natural question whether the combination of these two methods would yield even better parsers is a natural question.

The columns COMBINE in Table 4.10 show results of experiments in which supertag-

[10]We report parsing time. Exact runtime depend on implementation and hardware. We, therefore, give relative numbers so that the reader gets an impression of the magnitude.

ging and stacking features come from different sources, i.e., they were predicted by different tools.[11] For both parsers, the sequence labeler predicts the supertags, and the respective other parser provides the tree for the stacking features, i.e., $\text{COMBINE}^{\text{GB}}$ corresponds to $(\text{STAG}_{\text{SL}}{+}\text{STACK}_{\text{TB}})^{\text{GB}}$ and $\text{COMBINE}^{\text{TB}}$ is in fact $(\text{STAG}_{\text{SL}}{+}\text{STACK}_{\text{GB}})^{\text{TB}}$. We additionally provide MAX results from the best single source, i.e., MAX^{Y} is either $\text{STAG}^{\text{Y}}_{\text{SL}}$ or STACK^{Y}.

The results indicate that the combinations are better than the baseline. For most languages, the difference between the combination and the best single component is statistically insignificant, except Arabic, German, Hungarian, and English for $(\text{STAG}_{\text{SL}}{+}\text{STACK}_{\text{TB}})^{\text{GB}}$, and Arabic for $(\text{STAG}_{\text{SL}}{+}\text{STACK}_{\text{GB}})^{\text{TB}}$. The increment goes up to 0.51 in the case of Hungarian. On average however, the gains are marginal – graph-based parser's accuracy increases by 0.2 LAS, and the transition-based parser improves by 0.18 LAS Although these differences denote improvements, they are not nearly as high as the improvements over the baseline for the single components. Therefore, it depends on the actual dataset whether the architecture's complexity is worth the effort.

In Section 4.2, we argued that supertagging and stacking are similar, and the diversity of tools is the more important factor. The improvements achieved through their combination can also be interpreted along these lines: they are caused by using different tools rather than the fact that we are combining the two methods. In other words, it is like stacking onto two parsers instead of one.

4.5 Conclusion

This chapter started with the question of whether we can design a stacking architecture with a Level 0 model that does not predict well-formed trees. For this reason, we took a closer look at a different method for providing syntactic features for statistical dependency parsing – supertagging. Especially, we adopted the particular definition of supertagging from Ambati et al. (2014) and Ouchi et al. (2014).

We showed that supertagging is, in fact, a form of stacking. Although supertags carry less information than full trees, they improve dependency parsers to an equal amount. Moreover, using supertags in a parser that predicted them itself does not lead to improvements. This is in line with the findings of Surdeanu and Manning (2010) on stacking, of which supertagging is a variant. Therefore, while it is not so important which method is used, it is important to use different algorithms in these setups.

[11] We did experiments with combining supertags and stacking from the same Level 0 tool. However, since the features were derived from the same tree, the differences compared to only stacking were, as expected, negligible.

Our main conclusion is that stacking and supertagging are, in principle, interchangeable, and it depends on the application which of them is preferable. When speed is of the essence, supertagging allows for designing an ensemble architecture with two fast predictors: a Level 0 sequence labeler and a Level 1 greedy transition-based parser. The diversity between these two models is strong enough for the ensemble to benefit from it and improve the performance of the Level 1 parser. We applied this architecture in a cross-domain parsing scenario and demonstrated that it brings accuracy gains also in this setting.

When the accuracy is the priority, ensemble architectures with strong Level 1 models, i.e., graph-based or beam search transition-based parsers, achieve the highest accuracy. This accuracy can be further improved by combining stacking and supertagging, but only if different tools are used. These improvements come from the involvement of different tools rather than the combination of the two methods.

Finally, in the literature, stacking has also been proposed as a method to take advantage of parsers for other syntax frameworks. For example, Øvrelid et al. (2009b) showed an architecture in which a dependency parser was stacked on a slow grammar-driven LFG parser. In such a setting, supertagging might be preferable over stacking: rather than running the slow parser on every sentence in a stacking setup, it can be run once on some training data. A supertagger can then be trained on this data to provide syntactic information at a fraction of the cost.

Chapter 5

Cross-lingual vs. Monolingual Parsing in Low-Resource Scenarios

In the previous two chapters, we looked at ensemble dependency parsers within the context of practical applications and architectures' runtime. So far we left out the question of how the size of the training data influences the performance of such ensemble methods. Therefore, this chapter switches the focus from the topic of parsing runtime to the second methodological perspective addressed in this dissertation: **availability of training resources**.[1]

Statistical dependency parsers require manually annotated data, i.e., gold-standard treebanks, to learn from. Their performance strongly depends on the size of such treebanks. To illustrate this relationship, let us go back to Chapter 3, where we looked at combining diverse architectures to achieve robust parsing performance. There, Figure 3.4 provided performance analysis of three different parsers on a wide selection of treebanks. Among the treebanks for which the parsers performed poorly (with LAS less than the average) were the two smallest ones – Czech_CLTT and Hungarian – for which there is only 465 and 910 training sentences, respectively.[2] In both cases, ensemble methods brought significant improvements over single models, but those improvements were not enough to surpass the average 77 LAS threshold. In such a situation, the effort could be invested in building *cross-lingual* rather than *monolingual* architectures: against the background of Universal Dependency treebanks, one might hope that data available for other languages could be useful when the training treebank is small.

The central question we examine in this chapter is whether we can find cases where

[1] The content of this chapter is based on (Falenska and Çetinoğlu, 2017).

[2] We refer to Appendix A.2 for the treebank statistics.

cross-lingual parsers achieve better accuracy than monolingual parsers. As a related problem, we investigate the following case: when there is a new language to parse, with no treebank but with the chance to predict POS tags, should one pursue a cross-lingual parsing or invest in gold-standard annotation of at least a small corpus in the language itself? Before we develop and motivate these two questions further, we clarify the two concepts that have not been addressed in this dissertation yet – cross-lingual techniques and low-resource scenarios.

Cross-lingual techniques. *Cross-lingual methods* enable syntactic analysis for languages for which gold-standard treebanks do not exist. These methods involve transferring structural information from *source*, typically well-resourced languages, to low-resource *targets*. To enable such transfer, cross-lingual methods require consistent representations between source and target languages. This is why the development of the Universal Dependencies project (Nivre et al., 2016) facilitated and reinforced research into cross-lingual parsing techniques. Recall from Section 2.1.2 that the project aims at creating a framework for dependency treebank annotation that is consistent across languages: at the time of writing, it consists of 163 treebanks for 92 languages, and constantly grows.

The two predominant cross-lingual strategies are *annotation projection* (Hwa et al., 2005) and *direct model transfer* (Zeman and Resnik, 2008). Annotation projection involves transferring structures from the source language to the target through parallel sentences, i.e., sentences with translational equivalents in the target and source languages. Direct model transfer builds on a cross-linguistically shared categorization into POS tags and involves training a model on source sentences and applying it directly to the target data. Although both of the strategies rely on different principles, they share the same underlying initial question: *which source language should be used for a particular target to parse?* Answering this question and selecting a good source language is crucial for the performance of transfer methods (Bender, 2011). Usually, the decision is based on typological similarities between languages and can be performed manually or automatically. For example, the first approach to transferring dependency parsers was made by Zeman and Resnik (2008). The authors determined the relatedness of the involved languages manually. More recent approaches (Zhang and Barzilay, 2015; Aufrant et al., 2016, among others) use typological databases, such as the World Atlas of Language Structures (WALS) (Dryer and Haspelmath, 2013).

Both manual and automatic selection of source languages comes with challenges. Manual selection involves humans, who might overlook some of the less prominent language similarities. On the other hand, calculating language similarities automatically

is difficult because typological databases are sparse and usually come with poorly described languages (O'Horan et al., 2016). Moreover, typologically distant sources can turn out to be useful for parsing some targets (Lynn et al., 2014). Therefore, selecting only *one best* source language might not be the optimal approach for many target languages.

Multi-source transfer allows to bypass the problem of selecting only one best source language. It involves training separate models on several source languages and combining their predictions on the target side through ensemble methods. This method has been widely applied both to annotation projection (McDonald et al., 2011; Agić et al., 2016) and to direct model transfer (Zeman and Resnik, 2008; Rosa and Žabokrtský, 2015). In both cases, using few source languages has been shown to achieve higher parsing performance than using only one. Such popularity makes cross-lingual applications presumably the second most common use case of ensemble parsers after improving the performance of single monolingual models.

Low-resource scenarios. As we mentioned earlier, the performance of monolingual statistical dependency parsers, i.e., models that are trained and applied to the sentences in the same language, strongly depends on the size of the *training treebank*. When it comes to cross-lingual methods, their effectiveness depends on other resources. Annotation projection requires *parallel sentences* in source and target languages, while model transfer requires no gold-standard trees on the target side. However, target POS tags have to be predicted prior to parsing. Therefore, *corpora annotated with POS tags* have to exist to train the POS tagger.

There is a wide range of possible scenarios with respect to these three types of resources. For example, Universal Dependencies ver. 2.0 (Nivre et al., 2017) contains treebanks for *high-resource* languages, such as English or Italian. UD treebanks for these languages come with more than 10.000 sentences annotated with trees and POS tags, and the parallel collection OPUS (Tiedemann, 2012) has plenty of entries for them. Universal Dependencies ver. 2.0 covers also *low-resource languages*, such as North Sámi, Upper Sorbian, Buryat, or Kurmanji. These languages come with very small training sets (less than 20 sentences) and no development data.[3] When it comes to parallel corpora: three of these languages have only Linux distro translations in the parallel collection OPUS (Tiedemann, 2012), none are part of the 135-language Watchtower Corpus (Agić et al. (2016), but two have a few documents on the Watchtower website), and none are part of the 100-language Edinburgh Bible Corpus (Christodouloupoulos and Steedman, 2015).

[3]These four treebanks were designed as low-resource testing scenarios in the CoNLL 2017 UD shared task (Zeman et al., 2017).

Finally, no external POS-tagged corpora exist for them.

The latter low-resource languages are the focus of this chapter. Specifically, we put our attention at scenarios in which few manually annotated trees exist, but no parallel data is available. We find these scenarios particularly interesting for two reasons. First, they are relatively new because before the introduction of Universal Dependencies, treebanks with only a few manually-annotated trees were not common. Secondly, obtaining even low-quality automatic annotations for such low-resource languages might facilitate research for them, for example, by supporting the semi-automatic annotation process. Especially if we consider that the new and small UD treebanks are commonly created for languages for which no training data was available before.

Cross-lingual vs. monolingual low-resource parsing. Parsing low-resource languages is challenging. Especially when no parallel data is available, many standard cross-lingual parsing techniques are impossible to apply. Such techniques involve annotation projection, as well as treebank translation (Tiedemann and Agić, 2016), cross-lingual word clusters (Täckström et al., 2012), or word embeddings (Duong et al., 2015; Guo et al., 2015). The most commonly applied technique in this setting is called *multi-source delexicalized parsing* (Zeman and Resnik, 2008; Rosa and Žabokrtský, 2015). Figure 5.1 presents the central idea of the method. It involves training multiple source-side parsers without any lexical features, typically using only common POS tags. These single parsers are then applied to target *delexicalized sentences*, i.e., sentences with lexical features removed. Finally, the predicted trees are combined through blending.

Delexicalized parsing is a model transfer method, which means that it requires no gold-standard trees on the target side, but it needs POS tags to be predicted prior to parsing. This might be an issue, because, as we have noted above, low-resource languages commonly lack not only parallel data but also *external POS-tagged corpora*. In such cases, the small UD treebanks are the only sources of gold-standard POS tags. When the data for training POS taggers is small – as for many upcoming UD treebanks[4] – delexicalized methods might be affected by poor POS tagging accuracy. On the other hand, if there are *some* gold-standard trees, it is *possible* to train monolingual target parsers on them. Could there be cases in which such a low-resource monolingual parser is preferred over a delexicalized cross-lingual one?

Our goal is to investigate these questions systematically in low-resource settings and to find the conditions under which one strategy leads to better results than the other. Our

[4]For instance, at the time of writing, there are 21 upcoming treebank projects within Universal Dependencies http://universaldependencies.org/#upcoming-ud-treebanks.

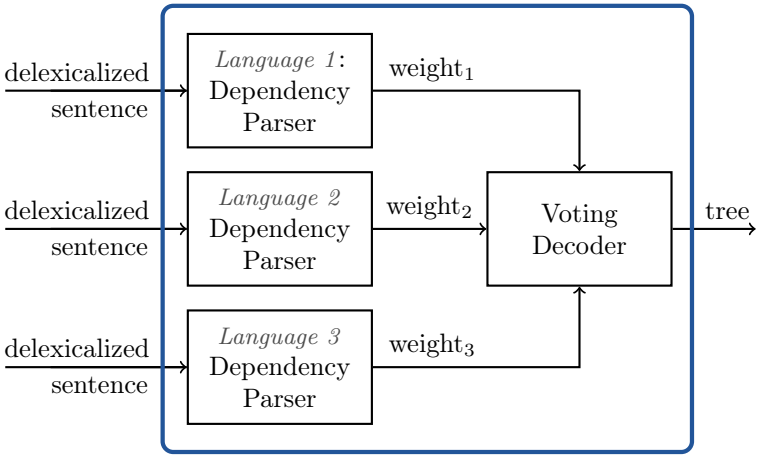


Figure 5.1: Delexicalized cross-lingual parsing with three source languages and weights.

conclusions can prove helpful in developing early parsing models for a new treebank. They can also help decide how to proceed when there is a large gold-standard POS tagged corpus but no trees exist (e.g., Echelmeyer et al. (2017) present a Middle High German corpus with 20,000 tokens of gold-standard POS, no trees, and no apparent parallel data). Moreover, our results can guide the planning of resource creation. While POS annotation can be relatively fast (Garrette and Baldridge, 2013), creating treebanks is costly (Zeman and Resnik, 2008; Souček et al., 2013). The decision to build a large POS annotated corpus vs. a small treebank in a limited time could depend on whether ensemble cross-lingual methods would work well for a particular target language.

5.1 Experimental Setup

This section specifies the experimental setup used for evaluation. We start by generalizing the low-resource scenarios that we have described above. Then, we explain the cross-lingual method that fits these settings the best. Finally, we give details of the datasets and tools used in our experiments.

5.1.1 Low-Resource Scenarios

Table 5.1. summarizes the two low-resource scenarios investigated in this chapter. In the first scenario (denoted **ExtPOS**), two different resources exist: a small training treebank and an external POS-annotated corpus. The corpus is larger than the treebank and allows for training a POS tagger with a good POS tagging accuracy. In the second scenario

	ExtPOS	TbPOS
Training treebank	very small to small	very small to small
Parallel data	no	no
POS gold-standard data	external	treebank
POS accuracy	good	varies

Table 5.1: Summary of the two low-resource scenarios addressed in our experiments.

(**TbPOS**), only the small training treebank exists. Thus, the gold-standard POS tags necessary to train a POS tagger must be extracted from the treebank. Therefore, while the POS accuracy is not affected by the treebank size in ExtPOS, it varies depending on the size of the treebank in TbPOS. Finally, in both of the scenarios we assume that no development data exists.

5.1.2 Cross-lingual Methods

The cross-lingual method that we apply in our experiments is *multi-source delexicalized parser transfer* (Zeman and Resnik, 2008; McDonald et al., 2011). As mentioned earlier, this method fits both of the experimental scenarios since it does not require any additional data, mainly parallel sentences. The idea is to train parsers on source treebanks using only non-lexical features, such as universal POS tags, and to apply them directly to sentences from the target language. We follow Rosa and Žabokrtský (2015) and Agić (2017) in combining predictions from different source parsers by weighted blending (we refer to Section 3.1.2 in Chapter 3 for a detailed description of the method). The weights of source languages employed in blending are based on language/treebank similarity measures. Since we assume that few annotated target trees exist, our testing scenarios differ from those addressed in the mentioned literature. Therefore, it is not clear what type of information, such as typological features, gold-standard POS tags, or gold-standard trees, should be considered when calculating the similarity between source and target languages. In order to ensure a fair comparison between cross-lingual and monolingual methods, we experiment with two different similarity measures and three different weighting techniques.

5.1.2.1 Similarity Measures

We employ two measures of similarity between languages known from the literature. The measures are then converted into weights (see the next section) and used for weighting

source languages during blending.

KL-POS. We use an adaptation of the KLCPOS-3 metric proposed by Rosa and Žabokrtský (2015). The metric is based on distributions of coarse POS tags in source and target corpora and is defined as the Kullback-Leibler divergence (Kullback and Leibler, 1951) between POS trigrams:

$$\text{KLCPOS-3}(tgt, src) = \sum_{t \in tgt} freq(t, tgt) \log \frac{freq(t, tgt)}{freq(t, src)}, \tag{5.1}$$

where tgt and src are target and source corpora, respectively; t is a trigram of POS tags; and the relative frequencies $freq$ of trigrams are estimated as:

$$freq(t, corpora) = \frac{count(t, corpora)}{\sum_{t' \in corpora} count(t', corpora)}. \tag{5.2}$$

For the KL divergence to be well-defined, the authors set the source count of each unseen trigram to 1.

In our experiments, we apply Rosa and Žabokrtský's (2015) metric to Universal Part-of-Speech (UPOS) tags and replace the original smoothing with Laplace smoothing with a learning rate of 0.01. We refer to this metric as **KL-POS**.

WALS. The previous metric uses corpora statistics but does not take into consideration the linguistic properties of sources and targets. Instead, Agić (2017) proposed to calculate language similarities by using the World Atlas of Language Structures (Dryer and Haspelmath, 2013). The dataset includes entries for 2,679 languages. The entries contain structural features divided into categories such as phonology, morphology, and syntax. We follow Agić (2017) and calculate the similarity of two languages as the Hamming distance between each language's WALS feature vectors, i.e., the number of positions at which the two vectors differ. For languages with no WALS entry, or for two languages with less than ten features in common, we do not define the **WALS** metric.

5.1.2.2 Source Weights

The conversion from similarities into weights is done in three ways using different types of information available in the training data. The first two methods come from Rosa and Žabokrtský (2015) and Agić (2017) and they use POS tags and WALS linguistic information. The third method makes use of small samples of target gold-standard trees.

R&Z15. Rosa and Žabokrtský (2015) calculated KL-POS between the *gold* POS tags of the source and target training data. Then, they convert the KL-POS negative measures of language similarity to positive weights by taking the fourth power of its inverted value KL-POS $^{-4}$. We use this method without any change and refer to it as **R&Z15**.

AGIC17. KL-POS and WALS have their strengths and drawbacks, namely the first metric needs more data for estimations, and the second one does not consider the treebank data at all. Therefore, Agić (2017) proposed to merge the two measures.[5] They first calculated KL-POS between the *gold* POS tags of the source training data and the *predicted* POS tags of the target development data. Then, they normalized KL-POS and WALS into probability distributions by applying a softmax function (see Equation (2.10)). Finally, they combined the two distributions through linear combination. The constants used in the combination were tuned on the development data. We follow all the described steps except for the last one. Instead, for simplicity, we keep the contribution of the two methods equal and combine them by simply summing their values.

TGTLAS. We introduce a third method for weighting source languages, which uses target gold-standard trees. We parse these target trees with the source delexicalized parsers. Then, we evaluate their quality with Labeled Attachment Score (LAS). The resulting LAS is used as weights in blending. The intuition is that with an extremely small number of trees, such weights might generalize better and catch more information about the syntactic similarity of languages than measures based on POS tags.

Since in preliminary experiments TGTLAS was achieving the best performance out of the three methods, we boost its behavior even further and introduce an additional tuning step. We use the LAS weights to rank all the source languages and take the n-best languages giving the best blended accuracy. This n is tuned for every treebank and every training size separately on the training data. We calculate LAS and tune on the same data because we assume that no development data exists for our test scenario.

5.1.3 Datasets

We perform experiments on the treebanks from the CoNLL 2017 UD shared task (Zeman et al., 2017), later released as Universal Dependencies ver. 2.0 (Nivre et al., 2017).

[5]Specifically, the authors worked with a setting where the language of text is unknown and combined three measures: KL-POS, WALS, and an output of a sentence-level language identifier.

	#tokens	#sentences
Upper Sorbian (hsb)	460	23
Kurmanji (kmr)	242	20
Buryat (bxr)	153	19
North Sámi (sme)	147	20

(a) Surprise languages, test-cases for EXTPOS.

	#tokens	#sentences
Latin (la)	18184	1334
Irish (ga)	13826	566
Ukrainian (uk)	1284	863
Uyghur (ug)	1662	100
Kazakh (kk)	529	31

(b) Small languages, test-cases for TBPOS.

Table 5.2: Number of tokens and sentences in low-resource test-case treebanks

As source languages, we take the 47 treebanks that are not domain-specific, i.e., for languages for which multiple UD treebanks exist, we take the canonical ones. We refer to Appendix A.2 for treebank statistics and ISO codes of these languages.

One of the objectives of the CoNLL 2017 UD shared task was to encourage participants to pursue cross-lingual approaches to dependency parsing. For this reason, the shared task introduced additional two testing scenarios with a goal of mimicking the challenges of parsing low-resource languages: *surprise* and *small* languages. We take these two groups of languages as our test-cases for EXTPOS and TBPOS settings.

Surprise languages. This scenario was called *surprise* in the CoNLL 2017 UD shared task because it consisted of four testing languages for which names were announced only one week before the test phase of the competition. Although at this point the names of these languages are known (i.e., Buryat, Kurmanji Kurdish, North Sámi, and Upper Sorbian), we keep calling them *surprise* for the sake of consistency.

Surprise languages mimicked the EXTPOS setting in which large POS training data exist and only a few gold-standard trees are available. Each of the treebanks came with roughly 20 gold-standard sentences for training and tuning parsing models (see Table 5.2a for statistics). The test sets were provided with POS tags predicted by a system trained on a dataset much larger than the training treebank.

Small languages. This group consisted of five languages for which no development data was available: Latin, Irish, Ukrainian, Kazakh, Uyghur. These languages came with small training treebanks (especially Kazakh and Uyghur, see Table 5.2b) and no additional POS-tagged data. They mimicked the TBPOS scenario where a few gold-standard trees exist, but POS tagging accuracy is poor.

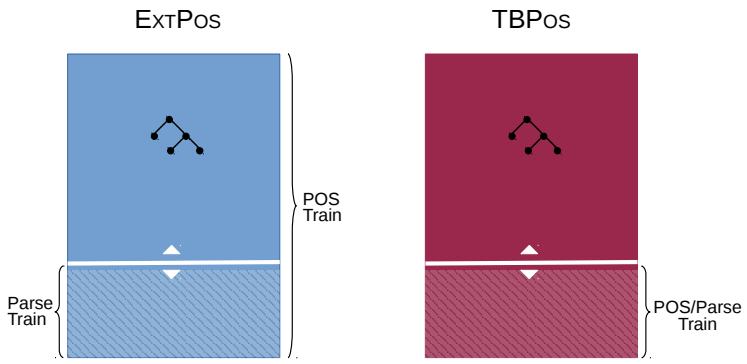


Figure 5.2: Schematic overview of training data used in the experimental setups.

Simulated low-resource languages. The nine target treebanks listed in Table 5.2 do not contain development sets. In order to not compromise our test sets, we set these languages aside as test cases and perform analysis on simulated low-resource languages instead. For this purpose, we use the subset of source treebanks that have a development set (41 languages). For each of them, we sample small training datasets, starting with only 100 tokens. Figure 5.2 illustrates how such sampling is performed for the two testing scenarios. For ExtPOS, POS taggers are trained on the whole dataset, and only training trees are sampled (bottom of the figure). For TbPOS, both sources of information come from the sampled part.

5.1.4 Tools and Evaluation

We use Labeled Attachment Score (LAS) as the evaluation metric and evaluate using the script provided by the CoNLL 2017 UD shared task organizers. Evaluation is always performed on the entire development sets.

Preprocessing. UPOS tags are used in all the experiments. We predict them with the CRF tagger MarMot (Mueller et al., 2013). In the ExtPOS setting, POS taggers are trained on the whole treebanks, while in TbPOS, training is done only on small samples of trees. We notice that for some of the languages, morphological features would be useful in parsing. However, in the cross-lingual scenario, some of the source languages would lack the morphological features that the target language would be expecting. Therefore, for a fair comparison, we exclude morphological information from both monolingual and cross-lingual parsing features.

Parsing. We perform parsing with the beam search transition-based parser IMSTRANS that was described in the previous chapter (Section 4.1.4). The parser uses the ARCSTANDARD system extended with a SWAP transition (Nivre, 2009), the lazy SWAP oracle (Nivre et al., 2009), and is trained with passive-aggressive update (Crammer et al., 2006).

We also experimented with the graph-based parser from mate-tools (Bohnet, 2010; Bohnet and Nivre, 2012) to test our hypotheses on a different parsing architecture. The parser uses Carreras's (2007) extension of Eisner's (1996) decoding algorithm to build projective parse trees. Then, it applies the non-projective approximation algorithm of McDonald and Pereira (2006) to recover non-projective dependencies. However, since in preliminary experiments the parsers achieved parallel results, we present only the results for the most efficient of them, i.e., the transition-based parser.

Finally, cross-lingual parsers and monolingual parsers for the TBPOS scenario are trained on gold-standard POS tags. For EXTPOS, monolingual parsers are trained on 5-fold jackknifed POS tags (we sample the small treebanks after performing jackknifing).

Blending. We combine outputs of delexicalized parsers through weighted blending. The weights are calculated according to the methods described in Section 5.1.2.2. To achieve labeled trees, we select the most frequent label across all of the single predictions. In presenting the experimental results, we refer to these ensemble cross-lingual models with their weighting scheme, namely **R&Z15**, **AGIC17**, and **TGTLAS**. Monolingual models trained on the target treebanks are referred to as **MONO**.

5.2 Cross-lingual Techniques vs. Monolingual Parsers

In this section, we compare cross-lingual methods with monolingual target parsers on the artificially created low-resource languages. The goal of the experiments is to find under what circumstances one method is better than the other.

5.2.1 The EXTPOS Scenario

We start by analyzing the results for the EXTPOS scenario. In this setting, POS tagging accuracy does not depend on the size of the parsing training data.

Average results. Figure 5.3 shows the average results for the EXTPOS scenario. We observe that cross-lingual methods using POS tags as their weight sources (AGIC17 and R&Z15) achieve, on average, 55% LAS. R&Z15 is slightly better than AGIC17, as big

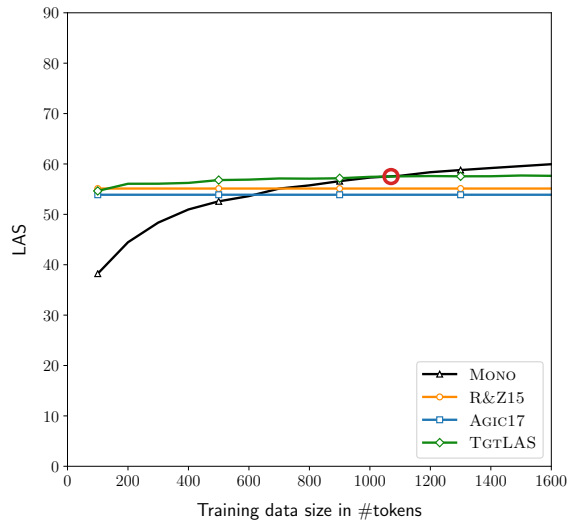


Figure 5.3: Average parsing accuracy for the EXTPOS scenario. The average *cutting point* is marked by the red circle.

gold-standard POS-tagged sets are available for this setting, and R&Z15 extracts gold-standard trigrams from the training data to calculate language similarities. However, both methods are surpassed by the monolingual model when gold-standard target trees are available for as little as 750 tokens (which corresponds to 40 sentences for the used treebanks). Creating weights from gold-standard trees (TGTLAS) instead of POS tags improves the blender (compared to AGIC17 and R&Z15). However, the improvement is modest, and the monolingual model trained on only 1,100 tokens (avg. 55 sentences) outperforms it. We find this number surprisingly small and we investigate it further.

Language-specific results. Instead of looking at averages, we analyze all target languages separately. We call the point (size, LAS) in which MONO surpasses TGTLAS (the red circle in Figure 5.3) a *cutting point*, and plot the cutting points for all of the target languages separately (Figure 5.4). The picture is more comprehensive than the average one. We note that most of the languages do not fall around the average cutting point of 1,100 tokens. Instead, they can be grouped into three clusters: 10 languages on the far left, for which a parser trained on even less than 500 tokens (avg. 25 sentences) is better than TGTLAS; 9 languages on the far right, for which even 3,400 tokens (avg. 176 sentences) are not enough to surpass the cross-lingual method; and those positioned in the middle section. In order to explain this distribution, we take a look at the characteristics of target languages.

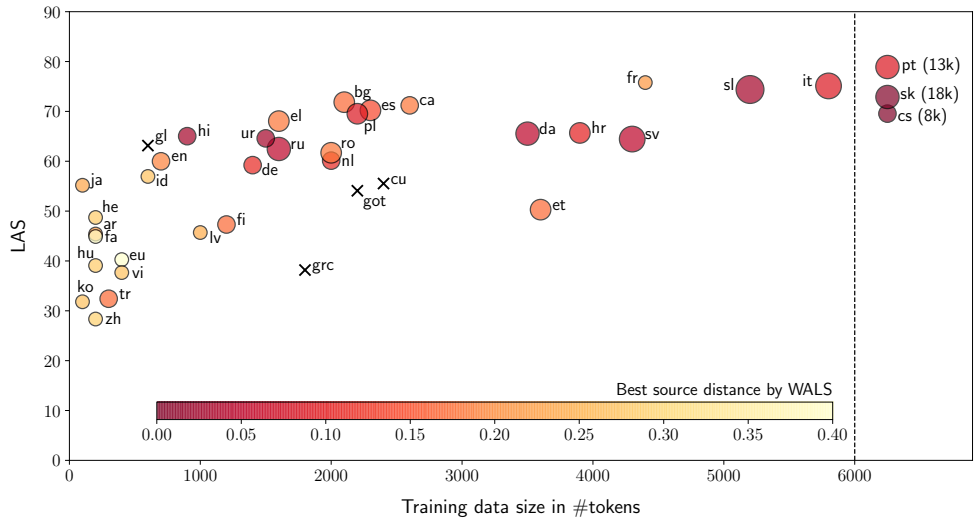


Figure 5.4: Cutting points for all of the artificial low-resource languages in the EXTPOS scenario. Points which fall far right are moved left and their corresponding training size is given in brackets. Cross (X) denotes no WALS entry. The circle size is proportional to the number of sources with WALS < 0.2

We use WALS to measure the distance between languages (we normalize WALS by the number of common features). For every target language, we represent by color its distance to the best source language according to WALS (the darker the circle, the more similar the best source is). The number of good sources (WALS < 0.2) is represented by size (the bigger the circle, the higher the number of good sources). For example, Korean's (ko) best source, according to WALS, is Urdu with a distance of 0.27, and its circle is light and small. In contrast, Slovenian (sl), for which five good sources exist (among which Ukrainian has the smallest distance of 0.03), is represented by a dark and big circle.

The tendency for the two border groups in Figure 5.4 is visible. The left group tends to have small and light circles, which means that there is no good source for them. When we look at the languages which fall into this group (like Arabic (ar) and Vietnamese (vi)), we see that they come from language families less represented among the source languages. For all of these languages, even an extremely small amount of annotated data is better than applying any of the transfer methods. The far-right circles in comparison are bigger and darker, which means that they fit well into the set of existing source languages. Indeed, many Slavic languages fall here. For such languages, transfer methods work very well and small amounts of gold trees can be used as source of weights, as in TGTLAS, rather than as training data for monolingual models.

5.2.2 The TBPOS Scenario

Before we summarize what we have learned about the relationship between cross- and monolingual methods, we move to the second TBPOS scenario in which both POS tagging and parsing accuracy depend on the treebank sizes.

Average results. Figure 5.5 shows the average results for TBPOS. As expected, the performance of parsers drops due to the lower POS accuracy. For example, in Figure 5.3, LAS is around 55 for 1000 tokens, whereas in Figure 5.5, it drops to around 36 for the same data size.

Other than the lower LAS scores, there are two major differences between these two plots. Recall that in this setting, POS taggers are trained on tags coming from the treebanks, and POS accuracy changes with the data size. That is why the AGIC17 and R&Z15 lines are not flat anymore. It is worth noting that the change in the size of POS-tagged training data influences the two methods in different ways. In R&Z15, with increased size, there are more gold-standard training data to calculate KL-POS. Hence, the method becomes more reliable. AGIC17, on the other hand, uses the same amount of POS tagged data all the time (i.e., the target development set), but its POS tagging accuracy increases with data size. Hence, it positively affects the KL-POS calculations. R&Z15 is slightly better than AGIC17, showing that the direct influence of the increasing number of gold-standard POS tags is higher than the influence of POS tagging accuracy. However, the difference between these two methods is once again very slight, and TGTLAS outperforms both of them for all data sizes.

As the second major difference to the EXTPOS scenario, we notice that MONO quickly outperforms all cross-lingual methods, i.e., the cutting point falls already at around 250 tokens (red circle on the Figure).

Language-specific results. Next, we look at language-specific results and show the breakdown of the cutting points by target language in Figure 5.6. The shades of the circles denote the POS accuracy (the darker the more accurate). In this case, the circles are placed in a smaller area with an upper limit of 3,100 tokens. This means that even less gold-standard target trees are needed in this scenario for the monolingual models to surpass cross-lingual approaches. Note that the source language similarity is not shown in this plot; yet, it still affects the underlying distribution of targets. Interestingly, in comparison to Figure 5.4, the relative placement of the languages is almost the same. The languages from less represented families, such as Korean (ko) or Arabic (ar), are on the

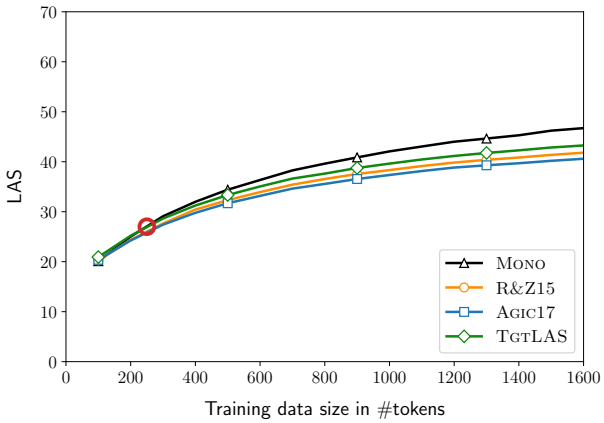


Figure 5.5: Average parsing accuracy in TbPOS scenario. The average *cutting point* is marked by the red dot.

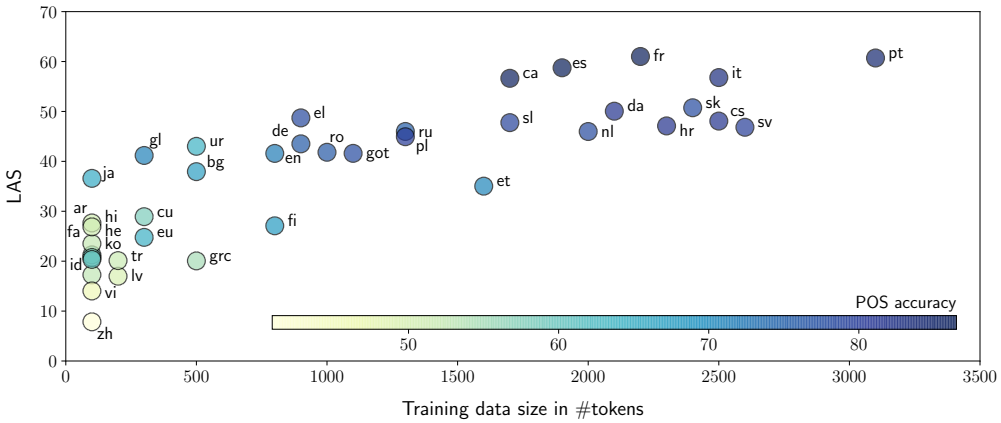


Figure 5.6: Cutting points for all artificial target languages in TbPOS scenario. The shades of the circles denote the POS accuracy (the darker, the more accurate).

left. And most of the Slavic languages are on the right. However, the dense distribution causes a more homogeneous scatter, making the groupings less visible.

5.2.3 Overall Picture

The results from both ExtPOS and TbPOS scenarios show that the choice between cross-lingual and monolingual approaches depends strongly on the existing resources. These resources include not only gold-standard target treebanks but also POS-tagged training sentences and treebanks for typologically similar source languages.

When training data for POS taggers is available (as in the ExtPOS scenario), good

sources are a precondition for the cross-lingual methods to work better than models trained on the target treebank. If the target belongs to an underrepresented language family, even a very small sample of gold-standard trees is enough for MONO to achieve better results. In contrast, if the low-resource target language is similar to many sources, cross-lingual techniques can help even if the target gold-standard data exists. In that case, gold-standard trees can be exploited similarly to TGTLAS as a source for blending weights. However, for all of our targets and regardless of existing sources, having a treebank bigger than 18,000 tokens is enough for a monolingual model to give higher accuracy than any of the cross-lingual methods.

When no other POS training data than the treebanks themselves exist (TBPOS), on average, monolingual models outperform cross-lingual methods even when trained on only a few gold-standard sentences. The only situation in which using other sources might help is visible when there are many similar sources and gold-standard trees for more than 1,000 tokens (avg. 50 sentences). In that case, the POS tagging accuracy can reach a reasonable 70%, and TGTLAS achieves comparable performance to MONO. However, having a treebank bigger than only 3,100 tokens (avg. 160 sentences) is already enough for MONO to work better for all the languages.

5.3 Application to Test Languages

The results from Section 5.2 can be interpreted as guidelines for deciding between cross-lingual and monolingual techniques. We first apply the guidelines to our test languages from Table 5.2 and hypothesize for which of these languages cross-lingual methods should be the choice. Then, the hypotheses are evaluated by comparing cross-lingual and monolingual techniques on the languages.

Application of guidelines. In the EXTPOS scenario, all the test case treebanks have less than 500 tokens (see Table 5.2a), and according to Figure 5.3, cross-lingual methods might be the right choice. Therefore, we look at the test languages closer and compare them with the breakdown in Figure 5.4. Buryat is a Mongolic language and does not have any close relatives among the source languages. For Kurmanji, there is only one similar language – Persian. Therefore, for both languages, the cutting point would likely occur quickly, and transfer methods would give little improvement over the monolingual model. In contrast, North Sámi is Finno-Ugric, and Finnish and Estonian should be good sources for it. Upper Sorbian is Slavic, and its family is well represented (e.g., by Czech or Polish). Therefore, cross-lingual methods should work very well for these two languages.

In the TBPOS scenario, most of the languages are much bigger than 3,100 tokens (see Table 5.2b), and for none of them cross-lingual techniques should help. Kazakh is smaller than the threshold of 1,000 tokens. Most probably, POS tagging accuracy for it would be poor, and MONO should be a better choice to overcome that. The only language for which cross-lingual methods might help is Uyghur, but it is a language for which not many good sources exist, the closest being Turkish. Most probably, MONO is a better choice also in this case.

Testing hypotheses. To test our hypotheses, we apply all the methods to the test languages and report results in Table 5.3. For languages not present in WALS (Kazakh, Uyghur), AGIC17 uses only KL-POS. We do not apply R&Z15 to the surprise languages since their POS tagging training data is not available[6]. We see that our intuition was right in 8 out of 9 cases. For all of the small languages, MONO performs better - even for Kazakh, where only 529 tokens (31 sentences) are available. As expected, in the case of surprise languages, Upper Sorbian and North Sámi gain from cross-lingual techniques the most – 9.22 and 6 points LAS, respectively, when compared to TGTLAS. For Kurmanji, MONO trained on only 242 tokens (20 sentences) gives 2.12 points LAS more than the best transfer method. Surprisingly, for Buryat, both cross-lingual methods outperform MONO, especially TGTLAS. We scrutinized this result and found out that the source parser that performs the best for this language is Basque. This is in-line with the results of Lynn et al. (2014), who showed that sometimes even typologically distant sources are useful for parsing particular languages. In these cases, methods which take advantage of the gold-standard trees, like TGTLAS, can help in finding such good sources.

5.4 Conclusion

In this chapter, we looked at ensemble dependency parsers from the perspective of the availability of training resources. We investigated low-resource parsing scenarios in which only a few gold-standard trees are available, POS tagging accuracy varies, and no reasonable amount of parallel data is available. To systematically compare cross-lingual and monolingual approaches and to observe under what circumstances one is more advantageous than the other, we created a simulation scenario with 41 low-resource languages and applied our findings to a set of 9 real low-resource targets.

[6]Their test sets were annotated via jackknifing by the CoNLL-17ST organizers to mimic the EXTPOS scenario.

	Statistics			Cross-lingual		
	#tokens	POS	MONO	R&Z15	AGIC17	TGTLAS
Latin (la)	18184	84.41	**41.02**	33.62	35.06	35.06
Irish (ga)	13826	89.99	**64.66**	41.75	42.17	44.54
Ukrainian (uk)	12846	88.58	**64.81**	62.20	58.46	63.67
Uyghur (ug)	1662	74.96	**34.81**	21.04	20.72	30.11
Kazakh (kk)	529	58.48	**26.55**	21.49	24.49	26.17

(a) Small languages

	Statistics			Cross-lingual		
	#tokens	POS	MONO	R&Z15	AGIC17	TGTLAS
Upper Sorbian (hsb)	460	90.30	49.59	–	57.09	**58.81**
Kurmanji (kmr)	242	90.04	**40.94**	–	38.82	38.17
Buryat (bxr)	153	84.12	28.06	–	29.07	**32.01**
North Sámi (sme)	147	86.81	29.80	–	33.44	**35.80**

(b) Surprise languages

Table 5.3: Results on the test languages. The best result for every language is marked in bold.

We found out that the small size of a target treebank should not be deceiving, as monolingual parsing can surpass cross-lingual methods even when trained on very few sentences. By analyzing the typological relations between source and target languages and the accuracy of POS tagging, it is possible to develop intuitions about what methods to apply to a new language. In general, our experiments show that if there are source languages close to the target, and if they are close enough, the target language benefits from cross-lingual parsing. Otherwise, monolingual parsers outperform cross-lingual ones even for small sizes of training data. For 8 out of 9 test languages, our findings hold.

In our experiments, we assumed specific constraints on existing resources, e.g., no parallel data between source and target languages. If some parallel data is available, other transfer parsing approaches can be applied. For instance, Agić et al. (2016) analyzed a related scenario where parallel data is available but there are no gold-standard trees. They showed that projecting POS and dependency annotations from multiple source languages outperforms single-best delexicalized parsing as well as blending. Whether such method would be a preferable choice when a small amount of gold-standard trees is

available is a question to explore. Finally, a scenario where both parallel data and few gold-standard trees are available was addressed by Guo et al. (2015). The authors showed that in such setting, the parallel data can be used to create word embeddings, and combining source and target language parsing models via cascading or multi-task learning can outperform single monolingual models. While both scenarios are beyond this chapter's scope, they should be taken into account in decision-making if some parallel data is available.

Chapter 6

Integrating Dependency Parsers in the Deep Contextualized Era

Up until now, we examined ensemble dependency parsers within the contexts of practical applications, parsing time, and availability of training resources. In this chapter, we turn to the third and final methodological perspective addressed in this dissertation: **the final accuracy of the ensemble**.[1] This perspective applies to the practical settings in which the main priority of the end user is to simply achieve as high final parsing accuracy as possible.

The deep contextualized era. Recall from Chapter 2 (Section 2.2.4) that to achieve accuracy on the level of today's state-of-the-art, we need to switch from the traditional dependency parsers, which employ discrete feature models, to neural models, which depend on continuous word representations. Since the first neural dependency parsers of Attardi (2006) and Chen and Manning (2014), such parsers have gradually replaced traditional models, they are being constantly developed, and are pushing the state-of-the-art in the dependency parsing field. One of the recent deep learning developments is based on encoding rich sentential context into word representations through techniques such as Bi-Directional Long Short-Term Memory (BiLSTM) (Hochreiter and Schmidhuber, 1997; Graves and Schmidhuber, 2005) or deep contextualized word embeddings (Peters et al., 2018; Devlin et al., 2019). Dependency parsers that use such techniques have set a new state-of-the-art for both graph-based and transition-based parsers (Kiperwasser and Goldberg, 2016b; Che et al., 2018). Moreover, they proved very effective in parsing

[1] The content of this chapter is based on (Falenska et al., 2020a).

competitions, for example, in the consecutive CoNLL shared tasks 2017 and 2018 on Universal Dependency parsing (Zeman et al., 2017, 2018). The two winning systems of these shared tasks, namely the submissions of Dozat et al. (2017) and Che et al. (2018), were based on BiLSTMs, and in the case of the latter, deep contextualized models.

Neural techniques vs. ensembles. The central question we examine in this chapter is whether the high accuracy of graph-based and transition-based neural parsers can be further improved through ensemble methods. On the one hand, in Chapter 3, we argued that ensemble methods are robust tools for improving parsing accuracy. Moreover, they bring improvements across diverse languages, in- and out-of-domain applications, and downstream tasks. On the other hand, there is one aspect that has to be considered, i.e., the accuracy of ensembles strongly depends on the diversity among the combined architectures. Including sentential context into representations of dependency parsers not only set a new state-of-the-art for both graph-based and transition-based parsers but also brought the two architectures closer. Kulmizev et al. (2019) showed that after including such representations, the average error profiles of graph-based and transition-based parsers converge, which might potentially reduce gains from combining them. However, the authors also noticed that the underlying trade-off between the parsing paradigms is still visible in their results. Thus, it is an open question *to what extent the differences between graph-based and transition-based parsers could still be leveraged when the deep representations underpin parsers of both paradigms*.

We start from the architecture of Kiperwasser and Goldberg (2016b), that the abovementioned systems of Dozat et al. (2017) and Che et al. (2018) were based on, and extend it with deep contextualized word embeddings. Through two well-established combination techniques – blending and stacking – we demonstrate that, on average, the differences between BiLSTM-based graph-based and transition-based models are reduced below the effect of different random seeds (Section 6.2). For example, blending still provides significant improvements. However, they do not come from the actual combination of different parsing paradigms but are a general artifact of neural training, where random seeds play a significant role. Interestingly, the diversity needed for a successful integration vanishes already with BiLSTM feature representations and does not change when deep contextualized embeddings are added (Section 6.3).

We further consider treebank-specific differences between graph-based and transition-based models (Section 6.4). Through a set of carefully designed experiments, we show that the graph-based parser has an advantage when parsing treebanks with a high ratio of right-headed dependencies. This advantage comes from globally trained Bi-

LSTM representations and can be exploited by the locally-trained transition-based parser through multi-task learning (Caruana, 1993). This combination improves the performance of the two parsing architectures and narrows the gap between them without requiring additional computational effort at parsing time.

6.1 Experimental Setup

This section specifies the experimental setup used for evaluation. We start by describing the parsing architectures and ensemble methods used in this chapter. Then, we give details of the treebanks and preprocessing steps. Finally, we explain how we evaluate the results and analyze the obtained models.

6.1.1 Parsing Model Architecture

Our graph- and transition-based parsers are based on Kiperwasser and Goldberg's (2016b) architectures (referred to as **K&G**). A detailed description and graphical illustration of the architectures are provided in Chapter 2 (Section 2.2.4.3). Here, we briefly recap them and describe the introduced changes.

We follow Kulmizev et al. (2019) and intentionally abstain from extensions such as Dozat and Manning's (2017) attention layer to keep our experimental setup simple. This enables us to carefully control for all relevant methodological aspects of the architectures. Our hypothesis is that adding more advanced mechanisms would resemble adding contextualized word embeddings, i.e., the overall performance improves but the picture regarding parser combination does not change. However, a thorough evaluation of this hypothesis is orthogonal to this work.

The described parsers are implemented with the DyNet library (Neubig et al., 2017). Table B.1 in Appendix B lists all of the used hyperparameters.

Deep contextualized word representations. The two most popular models of deep contextualized representations are ELMo (Peters et al., 2018) and BERT (Devlin et al., 2019). Both models have been used with dependency parsers, either for multi-lingual applications (Kondratyuk and Straka, 2019; Schuster et al., 2019) or to improve parsing accuracy (Che et al., 2018; Jawahar et al., 2018; Lim et al., 2018). Recently, Kulmizev et al. (2019) analyzed the influence of deep contextualized word representations on parsing architectures. They showed that for the K&G architecture, both ELMo and BERT give similar results, BERT being slightly ahead. Since the scope of our experiments is to

analyze the influence of contextualized embeddings and not to analyze the differences between different embedding models, we use ELMo, which at the time or writing is a more accessible model.

ELMo representations encode words within the context of an entire sentence. The representations are built from a linear combination of several layers of BiLSTMs pre-trained on a task of language modeling:

$$\text{ELMo}(x_{1:n}, i) = \gamma \sum_{j=1}^{L} s_j \text{BiLSTM}^j_{LM}(x_{1:n}, i) \tag{6.1}$$

We use pre-trained ELMo models provided by Che et al. (2018) and train task-specific parameters s_j and γ together with the parser. The final representations are combinations of $L = 3$ layers and have dimensionality equal to 1024.

Word representations. In both transition- and graph-based architectures, input tokens are represented in the same way. For a given sentence with words $[w_1, \ldots, w_n]$ and POS tags $[t_1, \ldots, t_n]$ each word representation x_i is built from concatenating: the embedding of the word, its POS tag, the BiLSTM character-based embedding, and the word's ELMo representation:

$$x_i = e(w_i) \circ e(t_i) \circ \text{BiLSTM}_{ch}(w_i) \circ \text{ELMo}(x_{1:n}, i) \tag{6.2}$$

Word embeddings are initialized with the pre-trained fastText vectors (Grave et al., 2018) and trained together with the model. The representations x_i are passed to the BiLSTM feature extractors and represented by a vector $\vec{x_i} \overleftarrow{} = \text{BiLSTM}(x_{1:n}, i)$.

Transition-based parser. Transition-based parsers operate on configurations. For every configuration consisting of a stack, a buffer, and the current set of arcs, the K&G parser builds a feature set from few crucial configuration items, concatenates their BiLSTM vectors, and passes them on to a Multi-layer Perceptron (MLP). The MLP scores all possible transitions, and the highest-scoring one is applied to proceed to the next configuration.

Our implementation (denoted **TB**) uses the ARCSTANDARD with SWAP transition system (Nivre, 2009) and can thus handle non-projective trees.[2] We use Nivre et al.'s (2009)

[2]We performed multiple experiments within the K&G architecture by differentiating transition-systems (e.g., removing SWAP, using the arc-hybrid system (Kuhlmann et al., 2011), and adding the dynamic oracle (Goldberg and Nivre, 2012, 2013)), graph-decoders (testing Eisner's (1996) algorithm), and word representations. In all the tested scenarios, the general picture was essentially the same. Therefore, we present results only for the best-performing configurations.

lazy SWAP oracle for training. We follow Shi et al. (2017a) and build the feature set from three vectors: the two top-most items of the stack and the first item on the buffer (denoted s_0, s_1, and b_0). Labels are predicted together with the transitions.

For analysis, we also use variants of the transition-based parser trained without BiLSTMs. In these cases, vectors x_i are passed directly to the MLP layer (similarly to Chen and Manning (2014)), and the implicit context encoded by the BiLSTMs is lost. We compensate for it by using Kiperwasser and Goldberg's (2016b) *extended* feature set $\{\vec{s_0}, \vec{s_1}, \vec{s_2}, \vec{b_0}, \vec{s_{0L}}, \vec{s_{0R}}, \vec{s_{1L}}, \vec{s_{1R}}, \vec{s_{2L}}, \vec{s_{2R}}, \vec{b_{0L}}\}$, where $._L$ and $._R$ denote the left- and rightmost child. The feature set adds the embedding information of additional tokens in the structural context of the parser state.

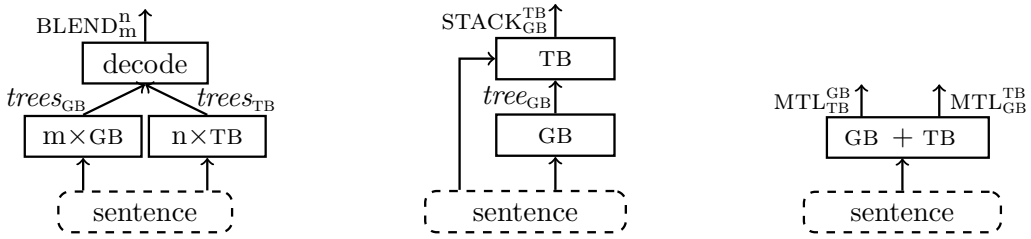
Graph-based parser. The graph-based K&G architecture used a first-order factorization. At parsing time, every pair of tokens $\langle x_i, x_j \rangle$ yields a feature set $\{\vec{x_i}, \vec{x_j}\}$. The BiLSTM representations of these features are concatenated and passed to an MLP to compute the score for an arc $x_i \xrightarrow{lbl} x_j$ for every possible dependency label lbl (unlike the original K&G implementation, we predict labels together with the arcs). To find the highest-scoring tree, we apply the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967). We denote this architecture **GB**.

In the experiments where GB is trained without BiLSTMs, we extend the feature set with surface features known from classic graph-based parsers, such as distance between heads and dependents, and words at a distance of 1 and 2 from heads and dependents (McDonald et al., 2005a).

6.1.2 Integration Methods

Parser combination approaches can be divided into two categories: methods that integrate base parsers during prediction and those which do the same during training. We use one well-established representative from each of the categories, i.e., blending and feature-based stacking. Additionally, as a tool for analysis, we combine the two parsers through multi-task learning. Blending and feature-based stacking are explained in detail in Chapter 3 (Section 3.1). Therefore, in this section, we recall the most important information about these methods and establish the notation used in the remainder of this chapter.

Blending. The method presented in Figure 6.1a involves running basic models in separation, combining their outputs into one graph, and employing a graph-based decoder to



(a) Blending **(b)** Stacking **(c)** Multi-task learning

Figure 6.1: Schematic illustration of the integration methods used in this chapter.

find the maximum spanning tree in the combined graph. In our implementation, we use the Chu-Liu-Edmonds algorithm to find the final tree. For every resulting arc, we select the most frequent label across all the labels previously assigned to it. Blending needs at least three basic models to apply the voting scheme. Therefore, we follow Kuncoro et al. (2016) and train multiple instances of models with random seeds, denoting $\textbf{BLEND}^{n}_{m}$ a combination of $m\times$GB and $n\times$TB parsers. For analysis, we vary the ratio of TB and GB models, while leaving the total number of models constant at 6 for a fair comparison.

Feature-based stacking. The method presented in Figure 6.1b involves running two parsers in sequence, such that the second (Level 1) parser can use the output of the first (Level 0) parser as features (denoted $\textbf{STACK}^{\textbf{LEVEL 1}}_{\textbf{LEVEL 0}}$). To generate training data for the Level 1 parser, we apply 10-fold cross-validation on the training sets with the Level 0 parser. Then, we follow Ouchi et al. (2014) and extract stacking features from the predictions of the Level 0 parser in the form of supertags. More precisely, for every word w_i we build its supertag by filling the template `label/hdir+hasLdep_hasRdep`, where `label` is the dependency relation, `hdir` denotes relative head direction, and `hasLdep/hasRdep` mark the presence of left/right dependents. Such supertags are then, similarly to POS tags, represented as embeddings and concatenated with other representations to build x_i. We made exploratory experiments to determine the choices of dimensionality and type of information encoded in the stacking representations on the English development data, and we left them unchanged for other languages.

Multi-task Learning (MTL). When analyzing BiLSTM representations, we combine the transition- and graph-based K&G parsers by sharing their BiLSTM representations (see

Figure 6.1c).[3] We keep feature extraction and MLP layers separate, and we do not enforce any agreement between the two decoders. In practice this means that training yields two parsers that can be applied independently: one transition-based (denoted $\text{MTL}^{\text{TB}}_{\text{GB}}$) and one graph-based ($\text{MTL}^{\text{GB}}_{\text{TB}}$).

We use a straightforward MTL training protocol: for every sentence, we calculate the BiLSTM representations $\vec{x_i}$ and collect all local losses from both tasks (TB and GB). Then the losses are summed, and the model parameters are updated through backpropagation. We note in passing that this training protocol leaves many options for improvements, such as adding weights to losses from different tasks (Shi et al., 2017b), sharing representations on different levels of BiLSTMs (Søgaard and Goldberg, 2016), or employing stack-propagation (Zhang and Weiss, 2016). We abstain from such extensions as they are orthogonal to the central points of our analysis.

6.1.3 Datasets and Preprocessing

We conduct experiments on a selection of thirteen treebanks from Universal Dependencies v2.4 (Nivre et al., 2019): Arabic (PADT), Basque (BDT), Chinese (GSD), English (EWT), Finnish (TDT), Hebrew (HTB), Hindi (HDTB), Italian (ISDT), Japanese (GSD), Korean (GSD), Russian (SynTagRus), Swedish (Talbanken), and Turkish (IMST). This selection was proposed by Kulmizev et al. (2019) and varies in terms of language family, domain, and amount of non-projective arcs. We refer to Appendix A.3 for treebank statistics.

We use automatically predicted universal POS tags in all of the experiments. The tags are assigned using a CRF tagger MARMOT (Mueller et al., 2013). We annotate the training sets via 5-fold jackknifing.

6.1.4 Evaluation and Analysis

We evaluate the experiments using Labeled Attachment Score (LAS).[4] We train our models for 30 epochs and select the best model based on development LAS. For the test sets results, we follow Reimers and Gurevych's (2018) recommendation and report averages and standard deviations from six models trained with different random seeds. We test for significance using the Wilcoxon rank-sum test with a p-value < 0.05.

[3] Shi et al. (2017b) used MTL in a similar way. They shared BiLSTM feature extractors between three parsers to speed up training time. However, they did not evaluate if the combination improved the performance of single models.

[4] The percentage of tokens that received the correct head and label.

	TB	$\text{STACK}^{\text{TB}}_{\text{GB}}$	GB	$\text{STACK}^{\text{GB}}_{\text{TB}}$
Arabic	82.59	82.59	82.95	83.02
Basque	79.96	80.35	82.21	81.71
Chinese	80.27	80.71	81.11	80.80
English	86.61	86.72	86.97	87.15
Finnish	86.75	86.38‡	87.16	87.47
Hebrew	85.57	85.94‡	86.55	86.62
Hindi	91.00	91.19	91.58	91.32‡
Italian	90.61	90.83	91.18	91.22
Japanese	93.47	93.32	93.34	93.39
Korean	82.17	81.95	82.99	82.62
Russian	90.30	90.60	90.90	90.83
Swedish	86.52	86.65	87.09	86.97
Turkish	63.94	64.18	66.19	66.03
average	84.60	84.72	85.40	85.32

Table 6.1: Average (from six runs) parsing results (LAS) on test sets for stacking. Corresponding standard deviations are provided in Table B.2 in Appendix B. ‡ marks statistical significance compared to single model baselines (p-value < 0.05).

An analysis is carried out on the development sets in order not to compromise the test sets. We follow Kulmizev et al. (2019) and sample the same number of sentences from every development set (484 sentences since this is the size of the smallest one). We then aggregate results from three models trained with different random seeds and present the combined results.

6.2 Diversity-Based Integration

We start by evaluating the two integration methods (STACK and BLEND) and by applying them to our transition- and graph-based parsers (TB and GB). Results are presented in Tables 6.1 and 6.2.

Average results. We first compare the average results of the two baseline models and the integration methods. In the case of stacking, we can notice in Table 6.1 that the average performance of the combined models is almost the same as that of the single ones. Small improvements are noticeable for $\text{STACK}^{\text{TB}}_{\text{GB}}$ vs. TB, but they are statistically significant only for one treebank, i.e., Hebrew. Comparing $\text{STACK}^{\text{GB}}_{\text{TB}}$ vs. GB, we even notice a

	Only TB		Only GB		TB and GB
	TB	BLEND_0^6	GB	BLEND_6^0	BLEND_3^3
Arabic	82.59	83.66	82.95	84.09	84.08
Basque	79.96	82.19	82.21	83.98	83.70
Chinese	80.27	82.31	81.11	82.83	83.01
English	86.61	87.85	86.97	87.95	88.11
Finnish	86.75	88.25	87.16	88.35	88.61
Hebrew	85.57	86.87	86.55	87.68	87.41
Hindi	91.00	91.79	91.58	92.16	92.05
Italian	90.61	91.56	91.18	91.93	91.95
Japanese	93.47	93.92	93.34	93.97	94.05
Korean	82.17	84.02	82.99	84.41	84.31
Russian	90.30	91.34	90.90	91.80	91.85
Swedish	86.52	88.10	87.09	88.49	88.67
Turkish	63.94	66.90	66.19	68.52	68.34
average	84.60	86.06	85.40	86.63	86.63

Table 6.2: Parsing results (LAS) on test sets for blending. Since blending already involves multiple models, we run it only once and do not test the results for significance.

small average drop of 0.08 LAS.

In the case of blending (see Table 6.2), the method provides strong improvements over single baselines (BLEND_0^6 vs. TB and BLEND_6^0 vs. GB). However, these improvements are not coming from integrating different paradigms, since BLEND_3^3 achieves the same average performance as BLEND_6^0, which uses only GB.

There are two possible explanations for the lack of average gains from integrating different parsing paradigms: either (1) in general, the neural models are simply not capable of benefiting from such combination, or (2) feature representations based on the BiLSTMs and the deep contextualized representations bring the architectures too close to each other for the integration to be beneficial. Section 6.3 investigates which of the architectural aspects are responsible for no benefits from the traditional combination methods.

Treebank-specific differences. We continue analyzing results presented in Tables 6.1 and 6.2 and take a closer look at differences on the treebank level. Comparing single baselines (TB vs. GB), we note that GB has a clear advantage over TB. It surpasses TB on twelve out of thirteen treebanks (all improvements are significant). We reproduce the analysis from Kulmizev et al. (2019) and confirm that this advantage is consistent across

arcs of different lengths, distances to root, and sentences with different sizes (we provide corresponding plots in Appendix B.2). Interestingly, the dominance of GB over TB significantly differs across treebanks and is especially prominent for more challenging ones, e.g., with small amounts of training data or a high level of non-projectivity. For instance, the largest difference of 2.25 LAS is visible for Basque, which is the treebank with the largest number of non-projective arcs, and Turkish, which has the smallest training dataset. Moreover, these are the treebanks where $\text{STACK}^{\text{TB}}_{\text{GB}}$ offers small improvements (0.39 LAS and 0.24 LAS, respectively), but both $\text{STACK}^{\text{GB}}_{\text{TB}}$ and BLEND^{3}_{3} cannot make use of the diversity in predictions of the two models and cause the accuracy to drop (observable by comparing $\text{STACK}^{\text{GB}}_{\text{TB}}$ vs. GB and BLEND^{3}_{3} vs. BLEND^{0}_{6}). As we saw in Chapter 4 (Table 4.8), in the case of non-neural parsers, a big gap between the performance of a strong graph-based model and a greedy transition-based model does not prevent the former from learning from the latter. Therefore, the questions arise where these treebank-specific differences come from and why integration methods cannot benefit from them. We address these questions in Section 6.4.

6.3 Parsing Architectures and Diversity

To investigate for what reasons there are no gains from the integration, we run ablation experiments and apply both blending and stacking on models trained with and without BiLSTMs and with and without ELMo representations.

6.3.1 Feature-based Stacking

We perform stacking with different types of Level 0 information. Apart from the standard way, in which TB is stacked on top of GB or vice versa (denoted O; for other) we carry out two types of control experiments: S (for self), where we stack a model on itself, and G (for gold), where gold-standard trees are used as Level 0 predictions.

Oracle experiments. Figure 6.2 displays results for stacking with different Level 0 information. We immediately see that scenario G, in which models are stacked on gold-standard trees, exhibits almost perfect performance. Regardless of the Level 1 parser and employment of BiLSTMs and ELMo, all models achieve accuracy higher than 95 LAS, proving that they are capable of learning from the stacking representations.

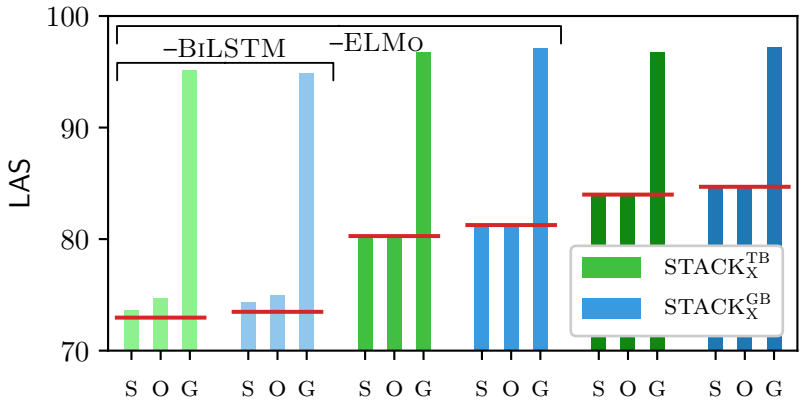


Figure 6.2: Average parsing accuracy (LAS over thirteen treebanks on development sets) for stacking with different types of Level 0 information: S – self, O – other, G – gold. Separate results for models trained with or without BiLSTMs and with or without ELMo. Red lines mark the average LAS of the single baseline models.

Influence of representations. Next, we consider the models which were trained without BiLSTMs and ELMo (left, lightest bars). Surprisingly, for both TB (green) and GB (blue), small improvements can be noticed in the self-application scenario S. Looking back at the equivalent experiments for non-neural models (see Chapter 4, Section 4.2.4), we notice that this was not the case and self-application used to lead to at most tiny improvements. One explanation for this might be the diversity of the neural models coming with random seeds – it used to be less prominent in their non-neural versions (Reimers and Gurevych, 2017) and thus less beneficial in stacking.

In scenario O, which combines models of different types, improvements are clearer. Both STACK$_{GB}^{TB}$ and STACK$_{TB}^{GB}$ surpass both of the single baselines, proving that integration is beneficial when BiLSTMs and ELMo are not used.

Considering the case where BiLSTMs are included (middle) changes the picture. Self-application behaves almost on par with stacking the parsers on each other. The only modest improvement (amounting to 0.18 LAS on average) occurs for STACK$_{GB}^{TB}$, but it is not enough to surpass a single GB baseline.

As expected, adding ELMo (right, darkest bars) results in significant improvements comparing to the models without the representations. However, these improvements do not impact stacking results, and the picture regarding the integration of the architectures stays the same.

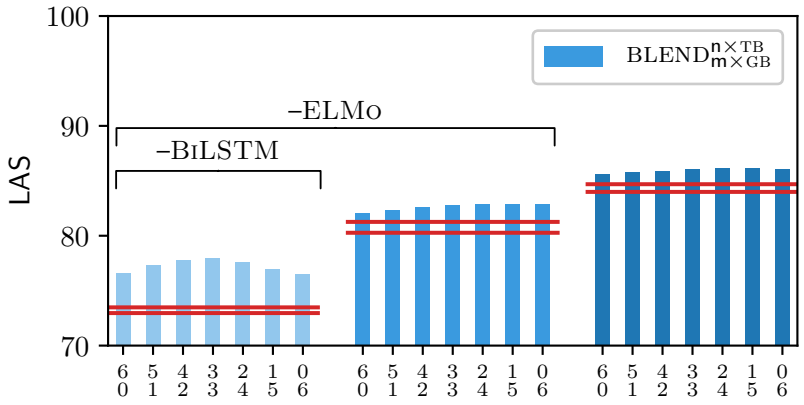


Figure 6.3: Average parsing accuracy (LAS over thirteen treebanks on development sets) for blending when models are trained with or without BiLSTMs and with or without ELMo. Red lines mark the average LAS of the single baseline models: top line – GB single model, bottom line – TB single model.

6.3.2 Blending

Figure 6.3 presents results for blending with different ratios of TB and GB models. We start by analyzing models trained without BiLSTMs and ELMo representations (left). We can observe a pattern we would expect from diverse models: (1) blending always improves over the baselines (signified by red lines); (2) combining models only of one sort (BLEND$^0_6$ or BLEND$^6_0$) yields lower scores than when we introduce more diversity into the combination; (3) the best result is obtained by BLEND$^3_3$, where the same number of TB and GB models is used.

For the models that use BiLSTMs (middle), the gains coming from blending are smaller. For example, BLEND$^6_0$ improves TB by 1.84 LAS, whereas the corresponding improvement when no BiLSTMs are used is 3.67 LAS. Interestingly, the models show a different pattern when it comes to diversity within the combination. The accuracy of the blend increases with the number of GB models. Although BLEND$^2_4$ achieves the highest accuracy, it surpasses BLEND$^0_6$ by only 0.05 LAS. This suggests that TB models do not bring enough diversity into the combination, and the accuracy of BLEND is mostly influenced by the performance of GB.

Finally, for models that use ELMo (right), improvements over single baselines are slightly smaller – BLEND$^0_6$ improves GB by 1.34 LAS comparing to 1.59 LAS when no ELMo is used. However, the picture regarding diversity is the same, and overall performance depends on the number of GB models and not on the diversity among combined paradigms.

To conclude, we showed that the performance of TB and GB models can be improved through the traditional diversity-based approaches as long as no BiLSTMs are used. Otherwise, the gains from combination methods decrease considerably. Adding ELMo representations improves the performance of both of the models but has almost no impact on the outcome of the integration.

6.4 Representations and Treebank-Specific Diversity

In the previous section, we found out that BiLSTMs mitigate average benefits from integration methods. One explanation might be that when both TB and GB use the same feature representations, the diversity between them is much smaller, thus reducing the gains that the models could draw from each other. However, when comparing treebank-specific results in Section 6.2, we noticed that the two baselines differ considerably in specific cases. We now investigate where these differences come from and if they can be beneficial.

6.4.1 Representation Analysis

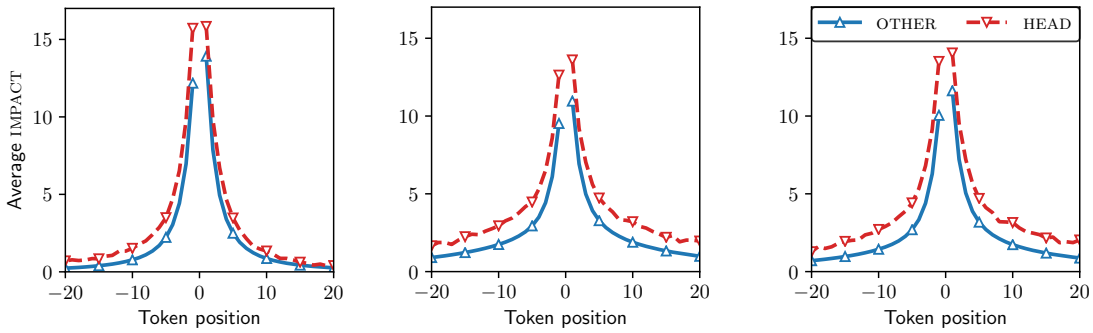
We first take a closer look at information encoded in representations learned by the transition- and graph-based models.

IMPACT metric. We follow Gaddy et al. (2018) and use derivatives to estimate how sensitive a particular part of the architecture is with respect to changes in input. Precisely, for every vector $\overrightarrow{x}$, we measure how it is influenced by every word representation x_i from the sentence. If the derivative of $\overrightarrow{x}$ with respect to x_i is high, then the word x_i has a strong influence on the vector. We compute the l_2-norm of the gradient of $\overrightarrow{x}$ with respect to x_i and normalize it by the sum of norms of all the words from the sentence calling this measure **IMPACT**:

$$\text{IMPACT}(\overrightarrow{x}, i) = 100 \times \frac{||\frac{\partial \overrightarrow{x}}{\partial x_i}||}{\sum_j ||\frac{\partial \overrightarrow{x}}{\partial x_j}||} \tag{6.3}$$

For every sentence from the development set and every vector $\overrightarrow{x_i}$, we calculate the impact of every representation x_j from the sentence on the vector $\overrightarrow{x_i}$. We bucket these impact values according to the distance between j and i.

Figure 6.4 shows the average impact of tokens at particular positions. We see the



(a) TB **(b)** GB **(c)** MTL

Figure 6.4: The average IMPACT of tokens on BiLSTM vectors trained with dependency parsers with respect to the token position.

same two general patterns as Gaddy et al. (2018): (1) closer words have larger effects on the representations, and (2) even words that are 15 or more positions away influence the vectors.

We continue our analysis and differentiate between the impact of correct heads (as looked up in the gold standard of the development set, red lines on the plot) from words that are not heads (blue). We see that the impact of heads is greater than the impact of other tokens. For example, in Figure 6.4a, tokens at a distance of -1 from $\overrightarrow{x_i}$ are responsible for around 15 percent of the change of $\overrightarrow{x_i}$ on average (among all words from the sentence), whereas this percentage drops to 13 percent for non-heads.

Transition-based parser. For representations trained with TB (Figure 6.4a), the difference in signals coming from heads and other tokens is bigger on the left side than on the right side (see, e.g., positions -15 and 15). De Lhoneux et al. (2019) provided an explanation for this. They showed that for greedy *locally-trained* models, the forward LSTM could be interpreted as rich history-based features while the backward LSTM could be thought of as look-ahead features. Since the information to the right mostly (i.e., except for the buffer front) comes from the backward LSTM, it contains, as in the case of standard look-ahead features, less structural relations.

Graph-based parser. Representations trained together with GB (Figure 6.4b) show a slightly different pattern. Compared to TB, the impact of heads is smaller for tokens closeby, but it also deteriorates slower. Since this model is *globally* trained, the influence of heads does not depend on the side – the plot is almost symmetrical (compare positions -20 and 20 in Figure 6.4b and Figure 6.4a), suggesting that representations encode as

much information about syntactic relations on the left as on the right.

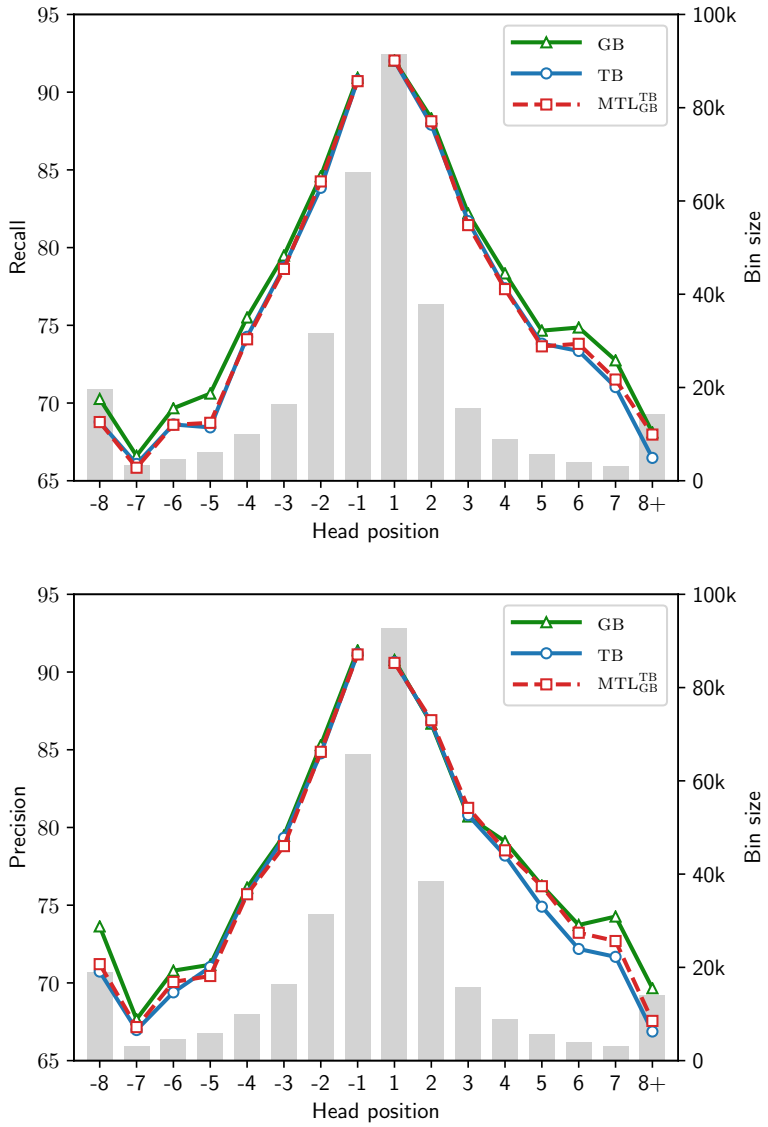


Figure 6.5: Dependency recall (top) and precision (bottom) relative to the head position on development sets.

6.4.2 BiLSTMs Integration

Next, we investigate whether the observed differences in the information encoded in BiLSTM representations can explain the advantage of GB over TB. We train new models where we share these intermediate representations between the two parsers through

MTL. Our hypothesis is that if the advantage of GB stems from global training and its influence on the representations, then MTL will re-balance the representations and, as a result, it will narrow the gap between the two models. We note in passing that although MTL is typically carried out on different tasks, often with different training sets, it is perfectly possible to consider graph-based and transition-based dependency parsing as two separate tasks trained on the same training set.

IMPACT analysis. Figure 6.4c displays the IMPACT statistics for MTL models. The plot shows that the BiLSTM representations draw on the advantages from both locally trained TB and globally trained GB – the distribution has a slightly stronger peak for closer words, as in TB, but flattens out more slowly, as in GB. This effect is particularly pronounced when comparing the far right (look-ahead) of TB with the MTL distribution, especially as heads become more influential.

Error analysis. To understand how the changes in representations influence the parsing performance, we break down the LAS by dependency length and head direction. Figure 6.5 shows the recall and precision of models with respect to the positions of the heads.[5]

First, we compare TB (blue) with GB (green) and observe that GB has a consistent advantage. However, when comparing recall and precision, we note an interesting difference. In terms of recall (top plot in Figure 6.5), the plot is symmetrical, and the advantage of GB is roughly the same for heads on the left and on the right side of the token. In terms of precision (bottom plot in Figure 6.5), both TB and GB behave identically for heads on the left, but the performance of TB drops faster for heads on positions 3 and more to the right.

Second, we analyze $\text{MTL}^{\text{TB}}_{\text{GB}}$ (red line). We notice that sharing representations with GB does not influence TB's recall or precision on heads on the left. However, for right-headed dependencies precision improves. The model catches up with GB's performance and starts deteriorating much later, for heads on positions 6 and more to the right.

Treebank specific improvements. Finally, we look at the treebank-specific accuracy of the MTL models. We start from the effects of MTL on GB (two right columns in Table 6.3). Sharing representations between the two architectures has a small influence on GB and,

[5]Dependency recall is defined as the percentage of correct predictions among gold standard arcs with head position p and precision is the percentage of correct predictions among all predicted arcs; the definition slightly differs from McDonald and Nivre (2007), who looked at absolute arc lengths.

	TB	$\text{MTL}^{\text{TB}}_{\text{GB}}$	GB	$\text{MTL}^{\text{GB}}_{\text{TB}}$
Arabic	82.59	82.55	82.95	82.99
Basque	79.96	80.90‡	82.21	82.55
Chinese	80.27	80.79	81.11	81.50
English	86.61	86.97	86.97	87.24‡
Finnish	86.75	87.09	87.16	87.52
Hebrew	85.57	85.90	86.55	86.68
Hindi	91.00	91.29‡	91.58	91.71
Italian	90.61	90.97	91.18	91.31
Japanese	93.47	93.45	93.34	93.47
Korean	82.17	82.36	82.99	83.12
Russian	90.30	90.58	90.90	91.11
Swedish	86.52	87.16‡	87.09	87.44
Turkish	63.94	65.22‡	66.19	65.88
average	84.60	85.02	85.40	85.58

Table 6.3: Average (from six runs) parsing results (LAS) on test sets. ‡ marks statistical significance compared to single model baselines (p-value < 0.05). Corresponding standard deviations are provided in Table B.3 in Appendix B.2.

on average, improves its performance by 0.18 LAS. Although for few treebanks bigger improvements can be seen, e.g., Chinese (0.39 LAS) or Swedish (0.35 LAS), none of them is statistically significant. Therefore, it is not clear if these improvements come from the actual combination of different parsing paradigms, or if MTL, in this case, acts as additional regularization, ultimately reducing overfitting during training.

In the case of TB, the average performance is improved through MTL by 0.42 LAS, with statistically significant differences for four treebanks. The biggest gains are visible for the treebanks, where the difference between TB and GB is the greatest, such as Basque (0.94 LAS). Interestingly, among the treebanks with the biggest improvements, we can notice Turkish (1.28 LAS) and Chinese (0.52 LAS), which are the two treebanks with the highest ratio of right-headed arcs (62.58% and 71.86%, respectively). This result is in line with the results of de Lhoneux et al. (2019), who demonstrated that backward LSTMs are especially important for head-final languages.

To conclude, we saw that the strong advantage of GB over TB stems from global training. The training increases the impact of tokens (far) to the right as compared to a locally trained TB model, which translates into an improved prediction of right-headed

dependencies. Thus the distance between the two models is treebank-related and can be reduced through integration methods such as MTL, especially when parsing more challenging treebanks.

6.5 Conclusion

In this chapter, we investigated ensemble dependency parsers with the focus on the final parsing accuracy. We started by explaining that to achieve accuracy on the level of today's state-of-the-art neural parsers have to be used. Especially parsers that employ techniques for incorporating sentential context into representations, such as BiLSTMs and deep contextualized embeddings, are successful at achieving robust parsing accuracy. Therefore, we asked under what circumstances such parsers could be further improved through ensemble methods.

We applied the two well-established ensemble techniques – feature-based stacking and blending – to integrate neural transition- and graph-based parsers based on Kiperwasser and Goldberg's (2016b) architectures. We found that when models use BiLSTMs, the diversity between them is on the level of different random seeds. Moreover, adding deep contextualized representations on top of BiLSTMs improves the performance of both parsers but does not change the picture regarding the integration.

In our setting, graph-based parsers have an advantage over transition-based parsers stemming from global training. In the case of pre-neural models, such an advantage was not limiting the strong models to learn from the weaker (cf., Section 4.3 in Chapter 4). We found that this is not the case for the neural models. Therefore, improving dependency parsers through combination methods is not as straightforward as it used to be. Such a combination has to take into consideration the specificity of the treebank and depends on whether accuracy or parsing time is the priority. The greatest gains in accuracy can be obtained by blending multiple graph-based models. However, this method comes with the cumbersome overhead of running multiple predictors at application time. When speed is essential, and the accuracy can be sacrificed (Gómez-Rodríguez et al., 2017), greedy transition-based parsers or even sequence labelers could be the preferable choices (Strzyz et al., 2019). In such cases, alternative integration approaches such as multi-task learning could boost the performance of locally-trained models without requiring additional computational effort at parsing time.

Chapter 7

The Utility of Structural Features in BiLSTM-based Dependency Parsers

In the previous chapter, we turned our attention to the final methodological perspective addressed in this dissertation: **the final accuracy of the system**.[1] We acknowledged that in order to move closer to the level of today's state-of-the-art, neural dependency parsers have to be used. Interestingly, we found out that this new neural era brings new challenges for integration methods. The methods are less straightforward to apply and offer less significant improvements than for their classical non-neural predecessors.

One of the conclusions of the previous chapter was that the predictions of neural transition-based and graph-based parsers lack diversity. Recall from Chapter 2 (Section 2.4), that such diversity between classical models used to be related to the trade-off between richness of feature representations on the one hand and exact or approximate search methods on the other. For example, graph-based parsers used to have an advantage on longer arcs, because they perform exhaustive search. And transition-based parsers used to be stronger on shorter arcs, where features drawn from structural context allowed them to make well-informed decisions.[2]

What changes when neural techniques, such as BiLSTMs, are added? Our BiLSTM-based graph-based parsers still perform global inference. However, our transition-based parsers *do not use* any conventional **structural features**. When Kiperwasser and Goldberg (2016b) proposed the architecture that our models are based on (referred to as **K&G** architecture), they achieved state-of-the-art performance without explicit information

[1] The content of this chapter is based on (Falenska and Kuhn, 2019).

[2] See Figure 7.1 for the concept of structural context. Details of the architectures will be described in Section 7.1.1.

about the structural context. The authors suggested that it is because the BiLSTM encoding is able to estimate the missing information from the given features and did not explore this issue further. Since the introduction of the K&G architecture BiLSTM-based parsers have become standard in the field.[3] Yet, it is an open question how much conventional structural context the BiLSTMs representations are actually able to capture *implicitly*. And whether employing structural features in the BiLSTM-based transition-based parsers would restore their advantage over the graph-based architectures and bring back the diversity among the methods.

Inspired by the work of Gaddy et al. (2018) on constituency parsing, we aim at understanding what type of information is captured by the internal representations of BiLSTM-based dependency parsers and how it translates into their impressive accuracy. As our starting point, we take the K&G architecture and extend it with a second-order decoder. We perform systematic analyses on nine languages using two different architectures (transition-based and graph-based) across two dimensions: with and without BiLSTM representations, and with and without features drawn from structural context. We demonstrate that structural features are useful for neural dependency parsers, but they become redundant when BiLSTMs are used (Section 7.2). It is because the BiLSTM representations trained together with dependency parsers capture a significant amount of complex syntactic relations (Section 7.3.1). We then carry out an extensive investigation of information flow in the parsing architectures and find that not only is the implicit structural context present in the BiLSTM-based parsing models but it is also more diverse than when encoded in explicit structural features (Section 7.3.2). Finally, we present results on ablated models to demonstrate the influence of structural information implicitly encoded in BiLSTM representations on the final parsing accuracy (Section 7.3.3).

7.1 Experimental Setup

This section describes the experimental setup used in this chapter. We start by recalling Kiperwasser and Goldberg's (2016b) architectures that we use for experimentation. Then, we give details of the datasets and preprocessing steps. Finally, we explain how we evaluate the results and analyze the obtained models.

[3]See results from the recent CoNLL 2018 shared task on dependency parsing (Zeman et al., 2018) for a comparison of various high-performing dependency parsers.

7.1.1 Parsing Model Architecture

We continue the experimental setup from the previous chapter and use graph- and transition-based parsers based on Kiperwasser and Goldberg's (2016b) architectures. We refer the reader to Chapter 2 (Section 2.2.4.3) for detailed description of these architectures. Here, we briefly summarize them and describe the introduced changes.

Since our goal in this chapter is to examine the information flow in the architecture, we simplify the architectures as much as possible. Therefore, we do not use the extensions added to the models in the previous chapter. More specifically, on the word representation level, we build token vectors only from the embeddings of words and their POS tags, and not character-based embeddings or deep contextualized representations. It means that for a given sentence with words $[w_1, \ldots, w_n]$ and part-of-speech tags $[t_1, \ldots, t_n]$ each word representation x_i is represented as:

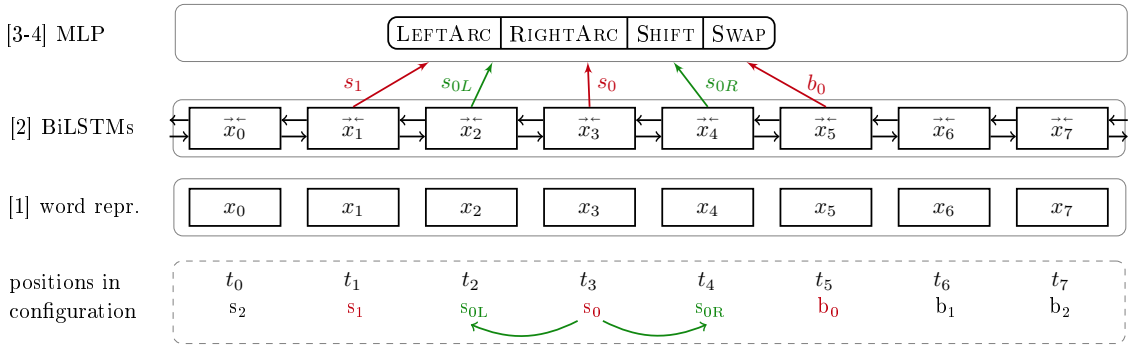
$$x_i = e(w_i) \circ e(t_i) \tag{7.1}$$

The embeddings are initialized randomly at training time and trained together with the model. This way, parsers do not depend on any pre-trained information, such as the fastText vectors (Grave et al., 2018).

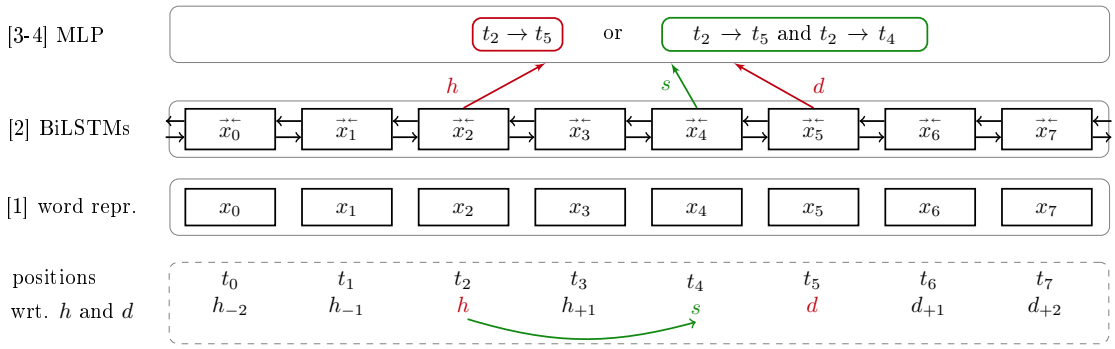
In Figure 7.1, we re-draw the K&G architectures (cf., Figure 2.19 in Chapter 2). We add information about the structural context for both transition- and graph-based parsers. For better readability, we omit level [3] of feature representations and combine it with level [4] (MLP).

In our experiments, we use two versions of transition-based and graph-based architectures: *simple*, which use no information coming from the structural context, and *extended*, which depend on structural features. Below we describe the two parsing architectures and differences between them.

Transition-based parser. Figure 7.1a illustrates the architecture of the basic K&G transition-based parser. The implementation we use (denoted **TB**) employs the ARC-STANDARD decoding algorithm extended with a SWAP transition (Nivre, 2009) to handle non-projective trees. We use the lazy SWAP oracle by Nivre et al. (2009) for training. Labels are predicted together with the transitions.
To compare the behavior of models with and without structural features, we experiment with two different feature sets:



(a) Transition-based parser; scoring transitions for a configuration $\langle$ [t_0, t_1, t_3], [t_5, t_6, t_7], $\{\langle t_3, t_2\rangle, \langle t_3, t_4\rangle\}\rangle$.



(b) Graph-based parser; scoring an arc $t_2 \rightarrow t_5$ or a second-order factor with two arcs $t_2 \rightarrow t_5$ and $t_2 \rightarrow t_4$.

Figure 7.1: Schematic illustration of the K&G architecture of BiLSTM-based neural dependency parsers (adaptation of Figure 2.19 from Chapter 2). Red arrows mark the basic feature sets, and green demonstrate how to extend them with features drawn from structural context.

TbMin is a simple architecture that does not use structural features. Shi et al. (2017a) showed that the feature set $\{\vec{s}_0, \vec{s}_1, \vec{b}_0\}$ (red arrows on the Figure 7.1a) is sufficient for the ArcStandard system, i.e., it suffers almost no loss in performance in comparison to larger feature sets but significantly outperforms a feature set built from only two vectors. Since we use an extended version of the ArcStandard system, we start from the same feature set. Later we analyze if the set could be further reduced.

TbExt is an extended architecture which employs additional structural features. We use the original extended feature set from Kiperwasser and Goldberg (2016b):

$\{\overrightarrow{s_0}, \overrightarrow{s_1}, \overrightarrow{s_2}, \overrightarrow{b_0}, \overrightarrow{s_{0L}}, \overrightarrow{s_{0R}}, \overrightarrow{s_{1L}}, \overrightarrow{s_{1R}}, \overrightarrow{s_{2L}}, \overrightarrow{s_{2R}}, \overrightarrow{b_{0L}}\}$, where $._L$ and $._R$ denote left- and rightmost child. The last seven vectors come from the structural context (green arrows on Figure 7.1a).

Graph-based parser. Figure 7.1b shows the K&G graph-based architecture. We experiment with two versions of this architecture:

GBMIN is the simple version that uses a feature set of two vectors $\{\overrightarrow{h}, \overrightarrow{d}\}$. At parsing time, every pair of words $\langle x_i, x_j \rangle$ yields a BiLSTM representation $\{\overrightarrow{x_i}, \overrightarrow{x_j}\}$ (red arrows in the figure), which is passed to MLP to compute the score for an arc $x_i \rightarrow x_j$. To find the highest scoring tree we apply Eisner's (1996) algorithm. We note in passing that although this decoding algorithm is restricted to projective trees, it has the advantage that it can be extended to incorporate non-local features while still maintaining exact search in polynomial time.[4]

GBSIBL is an extended architecture with additional information about the structural context. Specifically, we incorporate information about siblings $\overrightarrow{s}$ (green arrows in Figure 7.1b). The model follows the second-order model from McDonald and Pereira (2006) and decomposes the score of the tree into the sum of adjacent edge pair scores. We use the implementation of the second-order decoder from Zhang and Zhao (2015).

7.1.2 Data Sets and Preprocessing

We perform experiments on a selection of nine treebanks from Universal Dependencies ver. 2.0 (Nivre et al., 2017): Ancient Greek PROIEL, Arabic, Chinese, English, Finnish, Hebrew, Korean, Russian, and Swedish. This selection was proposed by Smith et al. (2018) as a sample of languages varying in language family, morphological complexity, and frequencies of non-projectivity (we refer to Appendix A.2 for treebank statistics). To these nine, we add the English Penn Treebank (PTB) (Marcus et al., 1993) converted to Stanford Dependencies as a standard parsing benchmark.[5] We use sections 2-21 for training, section 24 as a development set, and section 23 as a test set.

[4]Replacing Eisner's (1996) algorithm with the Chu-Liu-Edmonds's decoder (Chu and Liu, 1965; Edmonds, 1967) which can predict non-projective arcs, causes significant improvements only for the Ancient Greek treebank (1.02 LAS on test set).

[5]We use version 3.4.1 of the Stanford Parser from `http://nlp.stanford.edu/software/lex-parser.shtml`

We use automatically predicted universal POS tags in all the experiments. The tags are assigned using a CRF tagger MARMOT (Mueller et al., 2013). We annotate the training sets via 5-fold jackknifing.

7.1.3 Evaluation and Implementation Details

Experiments are evaluated using Labeled Attachment Score (LAS). We train models for 30 epochs and select the best model based on development LAS. We follow recommendations from Reimers and Gurevych (2018) and report averages and standard deviations from six models trained with different random seeds. We test for significance using the Wilcoxon rank-sum test with a p-value < 0.05.

The analysis is carried out on the development sets in order not to compromise the test sets. We present the results on the concatenation of all the development sets (one model per language). While the absolute numbers vary across languages, the general trends are consistent with the concatenation.

All the described parsers were implemented with the DyNet library (Neubig et al., 2017). We use the same hyperparameters as Kiperwasser and Goldberg (2016b) and summarize them in Table B.4 in Appendix B.

7.2 Structural Features and BiLSTMs

7.2.1 Simple vs. Extended Architectures

We start by evaluating the performance of our four models: TBMIN, TBEXT, GBMIN, and GBSIBL. The purpose of these experiments is to verify that the simple architectures will exploit BiLSTMs to compensate for the lack of additional structural features and achieve comparable accuracy to the extended ones.

Table 7.1 shows the accuracy of all the parsers. Comparing the simple (left) and extended architectures (right), we see that dropping the structural features does not hurt the performance, neither for transition-based nor graph-based parsers. In the case of graph-based models (GBMIN vs. GBSIBL), adding the second-order features to a BiLSTM-based parser improves the average performance slightly. However, the difference between these two models is significant only for two out of ten treebanks.

For the transition-based parser (TBMIN vs. TBEXT), a different effect can be noticed – additional features cause a significant loss in accuracy for seven out of ten treebanks. One possible explanation might be that TBEXT suffers more from error propagation than TBMIN. The parser is greedy, and after making the first mistake, it starts drawing features

	Transition-based		Graph-based	
	TBMIN	TBEXT	GBMIN	GBSIBL
Arabic	76.22	75.77	77.25	77.21
Chinese	73.28‡	72.17	74.47	74.95‡
English (UD)	81.85‡	80.50	82.53	82.65
English (PTB)	90.25	90.25	91.40	91.59
Finnish	72.51‡	71.47	74.37	74.44
Greek PROIEL	71.92‡	70.32	73.48	73.20
Hebrew	79.41‡	78.62	80.83	81.03
Korean	64.39	63.88	65.47	65.61
Russian	74.35‡	73.82	76.43	76.79‡
Swedish	80.11‡	78.80	81.22	81.42
average	76.43	75.56	77.74	77.89

Table 7.1: Average (from six runs) parsing results (LAS) on test sets. ‡ marks statistical significance (p-value < 0.05). Corresponding standard deviations are provided in Table B.5 in Appendix B.

from configurations that were not observed during training. Since the extended architecture uses more features than the simple one, the impact of the error propagation might be stronger. To investigate this effect further, we look at the average accuracy across all ten treebanks. Figure 7.2 displays accuracy relative to the dependency length in terms of recall and precision.[6] It confirms that the differences between simple and extended models are not restricted to arcs of particular lengths. However, the curves for TBMIN and TBEXT are almost parallel for the short arcs, but the performance of TBEXT deteriorates for the longer ones, which are more prone to error propagation (McDonald and Nivre, 2007).

7.2.2 Influence of BiLSTMs

We now investigate whether BiLSTMs are the reason for the simple models being able to compensate for the lack of features drawn from partial subtrees.

Transition-based parser. We train the transition-based parser in two settings: with and without BiLSTMs and with different feature sets. When no BiLSTMs are used, we pass

[6]Dependency recall is defined as the percentage of correct predictions among gold standard arcs of length l and precision as the percentage of correct predictions among all predicted arcs of length l (McDonald and Nivre, 2007).

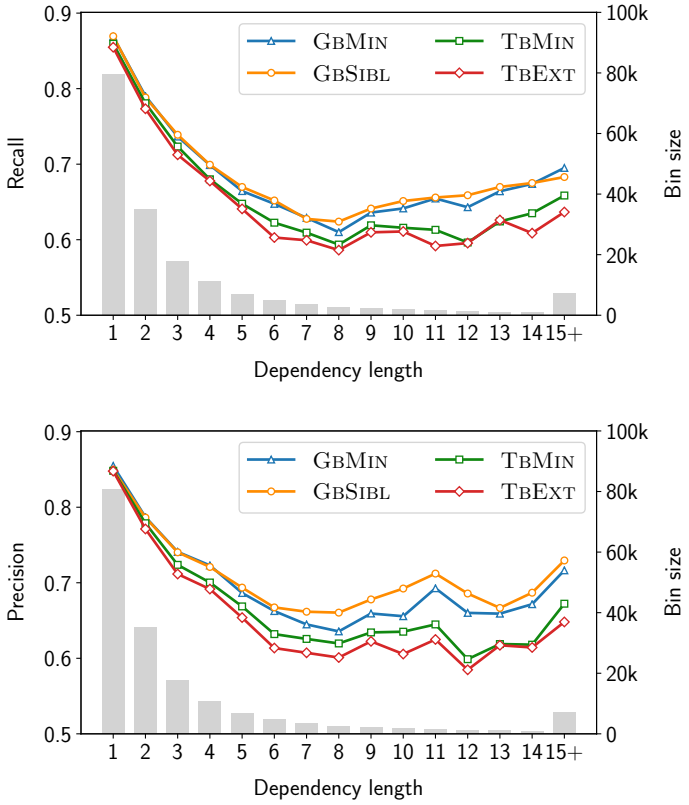


Figure 7.2: Dependency recall (top) and precision (bottom) relative to arc length on development sets; average results across all ten treebanks.

vectors x_i directly to the MLP layer following Chen and Manning (2014) (see Figure 2.16 in Chapter 2). We start with a feature set $\{s_0\}$ and consecutively add more vectors until we reach the full feature model of TbExt.

Figure 7.3 displays the accuracy of all the trained models. First of all, we notice that the models without BiLSTMs (light bars) benefit from structural features. The biggest gains in the average performance are visible after adding vectors s_{0L} (5.15 LAS) and s_{1R} (1.12 LAS). After adding s_{1R}, the average improvements become modest.

Adding the BiLSTM representations changes the picture (dark bars). First of all, as in the case of the basic ArcStandard system (Shi et al., 2017a), the feature set $\{\vec{s_0}, \vec{s_1}, \vec{b_0}\}$ is sufficient also when the transition Swap is used: (1) none of the other structural features are able to improve the performance of the parser, (2) but dropping b_0 causes a big drop of almost 6 LAS on average. Secondly, the parsers which use BiLSTMs always have a big advantage over the parsers which do not, regardless of the feature model used.

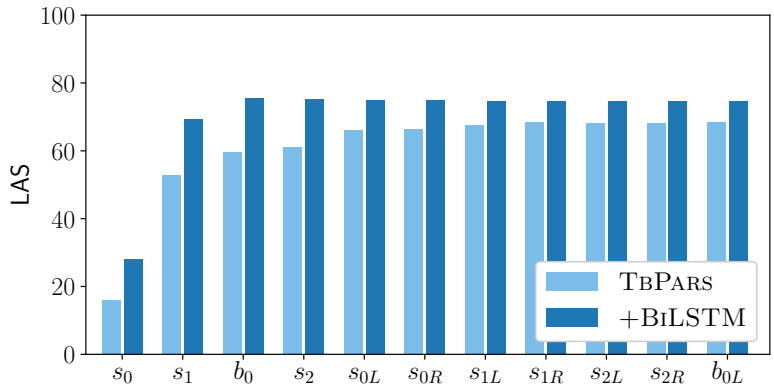


Figure 7.3: Parsing accuracy (average LAS over ten treebanks) with incremental extensions to the feature set of the transition-based parser.

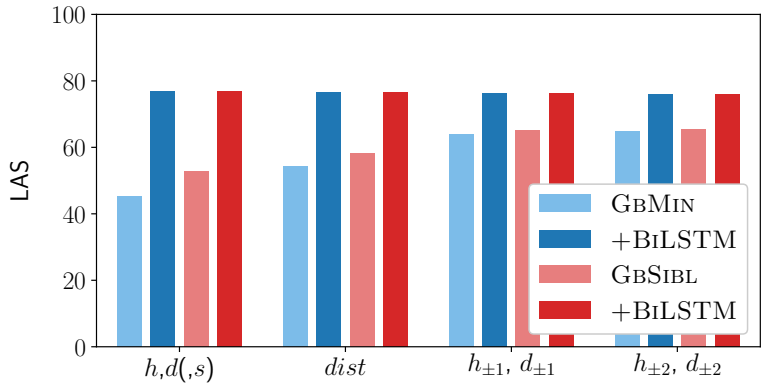


Figure 7.4: Parsing accuracy (average LAS over ten treebanks) with incremental extensions to the feature set of the graph-based parser.

Graph-based parser. We repeat similar experiments for the graph-based parser and train two models: GbMin and GbSibl with and without BiLSTMs. To ensure a fairer comparison with the models without BiLSTMs we expand the basic feature sets ($\{\overrightarrow{h}, \overrightarrow{d}\}$ and $\{\overrightarrow{h}, \overrightarrow{d}, \overrightarrow{s}\}$) with additional surface features known from classic graph-based parsers, such as distance between head and dependent ($dist$), words at a distance of 1 from heads and dependents ($h_{\pm1}$, $d_{\pm1}$) and at a distance ±2. We follow Wang and Chang (2016) and encode distance as randomly initialized embeddings.

Figure 7.4 displays the accuracy of all the trained models with incremental extensions to their feature sets. First of all, we see that surface features ($dist$, $h_{\pm1}$, $d_{\pm1}$, $h_{\pm2}$, $d_{\pm2}$) are beneficial for the models without BiLSTM representations (light bars). The improvements are visible for both parsers, with the smallest gains after adding $h_{\pm2}$, $d_{\pm2}$ vectors: on average 0.35 LAS for GbSibl and 0.83 LAS for GbMin.

As expected, adding BiLSTMs changes the picture. Since the representations capture surface context, they already contain a lot of information about words around heads and dependents, and adding features $h_{\pm1}$, $d_{\pm1}$ and $h_{\pm2}$, $d_{\pm2}$ does not influence the performance. Interestingly, introducing $dist$ is also redundant, which suggests that either BiLSTMs are aware of the distance between tokens, or they are not able to use this information in a meaningful way. Finally, even after adding all the surface features, the models which do not employ BiLSTMs are considerably behind the ones which do.

Comparing GBMIN (blue) with GBSIBL (red), we see that adding information about structural context through second-order features is beneficial when the BiLSTM are not used (light bars): the second-order GBSIBL has an advantage over GBMIN of 0.81 LAS even when both of the models use all the additional surface information (last group of bars on the plot). But this advantage is reduced to insignificant 0.07 LAS when the BiLSTMs are incorporated.

We conclude that, for both transition- and graph-based parsers, BiLSTMs not only compensate for the absence of structural features, but they also encode more information than provided by the manually designed feature sets.

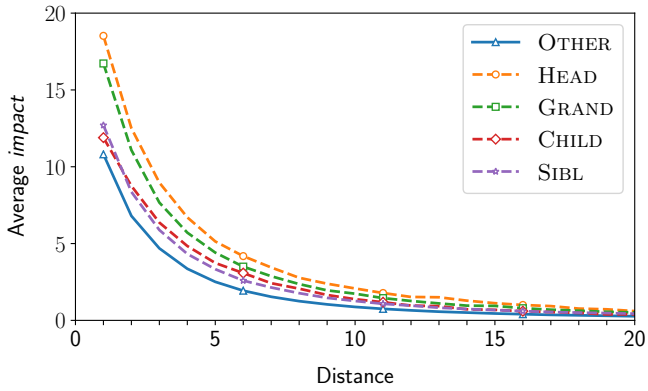
7.3 Implicit Structural Context

Now that we have established that structural features are indeed redundant for models that employ BiLSTMs, we examine the ways in which the simple parsing models (TBMIN and GBMIN) implicitly encode information about partial subtrees.

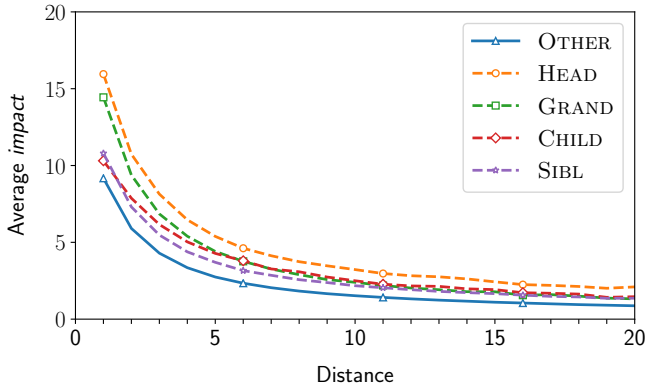
7.3.1 Structure and BiLSTM Representations

We start by looking at the BiLSTM representations. We know that the representations are capable of capturing syntactic relations when they are trained on a syntactically related task, e.g., number prediction task (Linzen et al., 2016). We evaluate how complicated these relations can be when the representations are trained together with a dependency parser.

To do so, we use our metric IMPACT from the previous chapter (Equation (6.3) in Chapter 6). The metric measures how every BiLSTM representation $\overrightarrow{x_i}$ is influenced by every word representation x_j from the sentence. Intuitively, IMPACT can be thought of as a percentage distributed over all words of the sentence – the higher the percentage of x_j the more it influenced the representation of $\overrightarrow{x_i}$.



(a) Transition-based parser (TBMIN)



(b) Graph-based parser (GBMIN)

Figure 7.5: The average impact of tokens on BiLSTM vectors trained with dependency parser with respect to the surface distance and the structural (gold-standard) relation between them.

For every sentence from the development set and every vector $\vec{x_i}$, we calculate the impact of every representation x_j from the sentence on the vector $\vec{x_i}$. We bucket these impact values according to the distance between the representation and the word. We then use the gold-standard trees to divide every bucket into five groups: correct heads of x_i, children (i.e., dependents) of x_i, grandparents (i.e., heads of heads), siblings, and other.

Figure 7.5 shows the average impact of tokens at particular positions. Similarly as shown in the previous chapter, even words 15 and more positions away have a non-zero effect on the BiLSTM vector. Interestingly, the impact of words which we know to be structurally close to x_i is higher. For example, for the transition-based parser (Figure 7.5a) at positions ± 5 an average impact is lower than 2.5%, children and siblings of x_i have a slightly higher impact, and the heads and grandparents around 5%. For the graph-based

parser (Figure 7.5b) the picture is similar with two noticeable differences. The impact of heads is much stronger for words 10 and more positions apart. But it is smaller than in the case of transition-based parser when the heads are next to x_i.

We conclude that the BiLSTMs are indeed influenced by the distance, but when trained with a dependency parser they also capture a significant amount of non-trivial syntactic relations.

7.3.2 Structure and Information Flow

Now that we know that the representations encode structural information we ask how this information influences the decisions of the parser.

First, we investigate how much structural information flows into the final layer of the network. When we look back at the architecture in Figure 7.1 we see that when the final MLP scores possible transitions or arcs it uses only feature vectors $\{\vec{s_0}, \vec{s_1}, \vec{b_0}\}$ or $\{\vec{h}, \vec{d}\}$. But thanks to the BiLSTMs the vectors encode information about other words from the sentence. We examine from which words the signal is the strongest when the parser makes the final decision.

We extend the definition of IMPACT to capture how a specific word representation x_i influences the final MLP score sc (we calculate the derivative of sc with respect to x_i). We parse every development sentence. For every predicted transition/arc we calculate how much its score sc was affected by every word from the sentence. We group impacts of words depending on their positions in the sentence.

Transition-based parser. For the transition-based parser, we group tokens according to their positions in the configuration. For example, for the decision in Figure 7.1a IMPACT$(sc, 1)$ would be grouped as s_1 and IMPACT$(sc, 4)$ as s_{0R}.

In Figure 7.6 we plot the 15 positions with the highest impact and the number of configurations they appear in (gray bars). As expected, s_0, s_1, and b_0 have the highest influence on the decision of the parser. The next two positions are s_{1R} and s_{0L}. Interestingly, these are the same positions which used as features caused the biggest gains in performance for the models which *did not* use BiLSTMs (see Figure 7.3). They are much less frequent than b_1 but when they are present the model is strongly influenced by them. After b_1 we can notice positions which are not part of the manually designed extended feature set of TBEXT, such as $s_{0\overline{L}}$ (left children of s_0 that are not the leftmost).

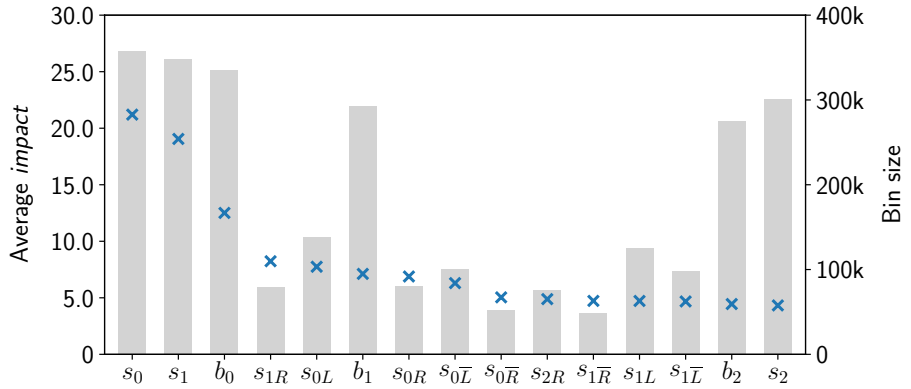


Figure 7.6: Positions with the highest average impact on the MLP scores (blue crosses) and their frequency (gray bars) for the transition-based parser (TBMIN). Positions depend on the configuration: $\cdot_{\overline{L}}$ marks left children that are not the leftmost, $\cdot_{\overline{R}}$ marks right children that are not the rightmost.

Graph-based parser. For the graph-based parser, we group tokens according to their position in the full predicted tree. We then bucket the impacts into: heads (h), dependents (d), children (c), i.e., dependents of dependents, siblings (s), and grandparents (g), i.e., heads of heads. Words which do not fall into any of these categories are grouped according to their surface distance from heads and dependents. For example, $h_{\overline{\pm 2}}$ are tokens two positions away from the head which do not act as dependent, child, sibling, or grandparent.

Figure 7.7 presents 10 positions with the highest impact and the number of arcs for which they are present (gray bars). As expected, heads and dependents have the highest impact on the scores of arcs, much higher than any of the other tokens. Interestingly, among the next three bins with the highest impact are children and siblings. Children are less frequent than structurally unrelated tokens at distance 1 ($h_{\overline{\pm 1}}$, $d_{\overline{\pm 1}}$), and much less frequent than $h_{\overline{\pm 2}}$ or $d_{\overline{\pm 2}}$ but they influence the final scores more. The interesting case is siblings – they not only have a strong average impact but they are also very frequent, suggesting that they are very important for the parsing accuracy.

The results above show that the implicit structural context is not only present in the models, but also more diverse than when incorporated through conventional explicit structural features.

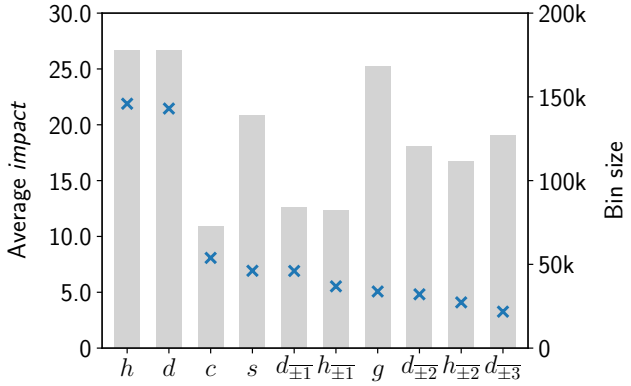


Figure 7.7: Positions with the highest impact on the MLP scores (blue crosses) and their frequency (gray bars) for the graph-based parser (GBMIN). Positions are: heads (h), dependents (d), children of d (c), siblings (s), grandparents (g), h,d$_{\overline{\pm i}}$ tokens at distance $\pm i$ from h or d which are none of h, d, c, s, or g.

7.3.3 Structure and Performance

Finally, we investigate if the implicit structural context is important for the performance of the parsers. To do so, we take tokens at structural positions with the highest impact and train new ablated models in which the information about these tokens is dropped from the BiLSTM layer. For example, while training an ablated model without s_{0L}, for every configuration we re-calculate all the BiLSTM vectors as if s_{0L} was not in the sentence. When there is more than one token at a specific position, for example $s_{0\overline{L}}$ or c (i.e., children of the dependent), we pick a random one to drop. That way every ablated model looses information about at most one word.

We note that several factors can be responsible for drops in performance of the ablated models. For example, the proposed augmentation distorts distance between tokens which might have an adverse impact on the trained representations. Therefore, in the following comparative analysis we interpret the obtained drops as an approximation of how much particular tokens influence the performance of the models.

Transition-based parser. Figure 7.8 presents the drops in the parsing performance for the ablated models.[7] First of all, removing the vectors $\{\overrightarrow{s_0}, \overrightarrow{s_1}, \overrightarrow{b_0}\}$ (marked in green on the plot) only from the BiLSTM layer (although they are still used as features) causes visible drops in performance. One explanation might be that when the vector $\overrightarrow{s_0}$ is re-

[7] It is worth noting that not all of the models suffer from the ablation. For example, dropping vectors s_{2R} causes almost no harm. This suggests that re-calculating the representations multiple times does not have a strong negative effect on training.

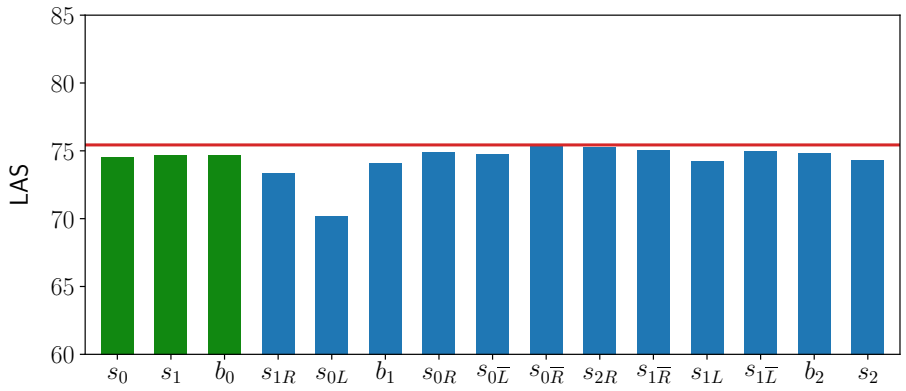


Figure 7.8: The performance drops of the transition-based parser (TBMIN) when tokens at particular positions are removed from the BiLSTM encoding. The red line marks average LAS of uninterrupted model. Feature set of the model is highlighted in green.

calculated without knowledge of s_1 the model loses information about the distance between them. Secondly, we can notice that other drops depend on both the impact and frequency of positions. The biggest declines are visible after removing s_{0L} and s_{1R} – precisely the positions which we found to have the highest impact on the parsing decisions. Interestingly, the positions which were not a part of the TBEXT feature set, such as $s_{0\overline{L}}$ or $s_{1\overline{R}}$, although not frequent, are important for the performance.

Graph-based parser. Corresponding results for the graph-based parser are presented in Figure 7.9 (we use gold-standard trees as the source of information about structural relations between tokens). The biggest drop can be observed for ablated models without siblings. Clearly, information coming from these tokens implicitly into MLP is very important for the final parsing accuracy. The next two biggest drops are caused by lack of children and grandparents. As we showed in Figure 7.7, children, although less frequent, have a stronger impact on the decision of the parser. But dropping grandparents also significantly harms the models.

We conclude that information about partial subtrees is not only present when the parser makes final decisions but also strongly influences these decisions. Additionally, the deteriorated accuracy of the ablated models shows that the implicit structural context cannot be easily compensated for.

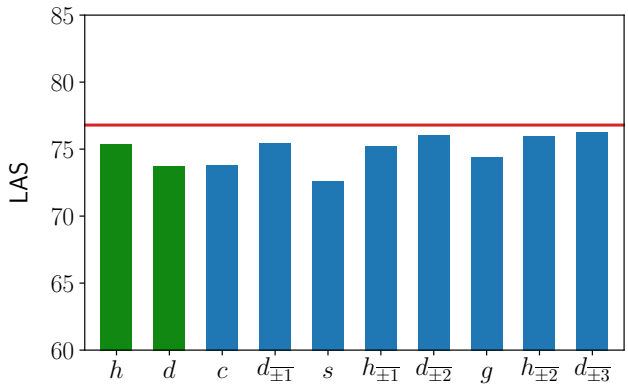


Figure 7.9: The performance drops of the graph-based parser (GBMIN) when tokens at particular positions are removed from the BiLSTM encoding. The red line marks average LAS of uninterrupted model. Feature set of the model is highlighted in green.

7.4 Conclusion

In this chapter, we examined the influence of BiLSTMs on the neural transition- and graph-based parsing architectures. We saw that the BiLSTM-based parsers can compensate for the lack of traditional structural features. Specifically, the features drawn from partial subtrees become redundant because the parsing models encode them implicitly.

The main advantage of BiLSTMs comes with their ability to capture not only surface but also syntactic relations. When the representations are trained together with a parser, they encode structurally-advanced relations such as heads, children, or even siblings and grandparents. This structural information is then passed directly (through feature vectors) and indirectly (through BiLSTMs encoding) to the multi-layer perceptron and is used for scoring transitions and arcs. Finally, the implicit structural information is important for the final parsing decisions: dropping it in ablated models causes their performance to deteriorate.

Using BiLSTMs in parsing architectures has strong repercussions not only on the parsers themselves but also on ensemble methods. The lack of diversity in predictions of transition- and graph-based models, that we saw in the previous chapter, is a result of blurring the differences between the two architectures. Both architectures use the same word representations. We showed that these representations trained together with the parsers, capture syntactic relations in a similar way. Moreover, the transition-based parser does not incorporate structural features through the feature set, as it used to before the introduction of the neural enhancements. And the graph-based parser makes use of far away surface tokens but also structurally related words. As a result, it has a strong

advantage that the locally-trained transition-based model cannot make up for.

The introduction of BiLSTMs into dependency parsers had another consequence, i.e., it enabled the use of exact search algorithms for transition-based parsers (Shi et al., 2017a; Gómez-Rodríguez et al., 2018). Therefore, it is a venue to explore if the error profiles of such parsers are even less distinguishable from the graph-based outputs. Or if the global-training, in this case, re-balances the errors of the two architectures and restores the benefits from combining the two approaches.

Chapter 8

Conclusion

This dissertation had its main focus on dependency parsing, the Natural Language Processing task of assigning a grammatical structure to a given sentence. We argued that in practical applications, parsing is rarely the end goal. It is more commonly one of the processing steps which supports downstream tasks, as it resolves some of the syntactic ambiguities naturally occurring in language.

Indeed, human language is highly ambiguous, and automatically deciding on the best syntactic structure for a given sentence can be extremely challenging. As a result, even state-of-the-art dependency parsers are far from being perfect and still make many mistakes. Ensemble methods reduce such parsing errors by performing the same task with a number of parsers and finding a consensus among their predictions.

This dissertation started from the assumption that ensemble methods, although powerful, are commonly overlooked in practical applications. We motivated this assumption in Chapter 3. We argued that, due to the long tradition of parsing shared tasks, ensembles are mostly associated with boosting parsing accuracy. Moreover, since ensembles come with higher complexity than single models, they are regarded as solutions mostly for experts. Countering this assumption, the goal of this dissertation was to provide a better understanding of ensemble dependency parsers and answer methodological questions that arise when applying them in real-world scenarios, such as processing web data or low-resource languages. In the following section, we summarize our main contributions.

8.1 Contributions

In Section 1.1, we defined the three core questions that have to be addressed when designing an ensemble parsing architecture. We now go back to these questions and summarize

what we have learned about ensemble methods from the methodological perspectives of parsing time, availability of training data, and the final accuracy of the system.

How fast does my system have to be? Ensembles provide robust results compared to single models. However, since they involve multiple predictors, they come with higher complexity and parsing time.

Chapter 3 provided an overview of practical applications of ensemble parsers. We discussed a number of use cases in which the size of the parsing architecture is not a bottleneck. For example, in semi-automatic treebank annotation, the automatic parsing step can take longer if, in return, it produces better results and decreases the necessary manual workload.

On the other hand, in large-scale applications such as processing web data, the speed of the parsers is essential. In such applications, lighter and faster ensemble parsers are preferred. We demonstrated that lighter ensembles can be built from individual components different from dependency parsers. Specifically, we looked into two independently introduced methods – feature-based stacking and supertagging. Feature-based stacking is a training-time integration technique that requires two parsers. Supertagging, as a method for providing syntactic features for a statistical dependency parser, typically involves a sequence labeler and a parser. We hypothesized and demonstrated that supertagging is a form of stacking. Although supertags do not carry as much information as full trees, the theoretical differences between the two methods have no impact in practice. Thus, the decision of which of the two methods to select depends on the specificity of the concrete application. Especially when speed is crucial, supertagging allows for the design of an ensemble architecture with two fast predictors: a sequence labeler and a greedy transition-based parser.

How much training data do I have? One of the first questions that must be addressed when designing a data-driven NLP architecture concerns the availability of training resources. The performance of dependency parsers strongly depends on the number of manually annotated dependency trees. When a gold-standard treebank is not available, two options have to be considered: creating a new treebank or pursuing ensemble cross-lingual techniques. The choice between these two options is dictated by the availability of other resources, such as parallel translations in the target and source languages. We demonstrated that the same options have to be taken into consideration even if few gold-standard trees exist. Moreover, in such a setting, a third option becomes available – training a monolingual target parser – which makes designing the parsing architecture

even more complex.

In this dissertation, we focused on truly low-resource scenarios, for which there is no parallel data and only a very small gold-standard treebank. Such scenarios have become more and more salient with the growing scope of the Universal Dependencies project. Moreover, they represent a challenge for processing tasks, given that many cross-lingual techniques depend on the existence of parallel data.

We compared two approaches that fit into the given low-resource constraints: (1) an ensemble cross-lingual technique called multi-source delexicalized parsing, and (2) monolingual parsers trained on small treebanks. We developed guidelines on how to select one method over the other depending on the typological features of the language and the availability of corpora annotated with POS tags. We showed that the ensemble cross-lingual techniques should only be pursued if treebanks for typologically similar source languages exist, and high-quality POS tags can be predicted. Otherwise, monolingual approaches are preferable, especially considering that monolingual ensembles can provide accuracy gains even when the training treebanks are small.

How to achieve the best possible accuracy for my application? Nowadays, to achieve state-of-the-art accuracy, the standard choice is to resort to deep learning techniques. In this dissertation, we investigated two of such neural enhancements: feature extractors based on BiLSTMs and deep contextualized embeddings. We found that the performance of dependency parsers that employ such techniques can be further improved through ensemble methods. However, different factors have to be considered when designing ensemble architectures for neural models than for traditional parsers.

In the pre-neural era, ensembles were the most efficient when they involved both transition- and graph-based parsers. These two types of architectures used to make complementary mistakes stemming from their inherent differences. Ensemble methods would benefit from these differences and achieve performance higher than that of the individual models.

For BiLSTM-based neural parsers, the diversity between transition- and graph-based models drops to the level of different random seeds. The explanation comes from the nature of the BiLSTM representations that underpin both parsing paradigms. These representations implicitly encode rich structural features when trained with transition-based as well as graph-based parsers. As a result, the error profiles of the two models converge. Since the graph-based parser still has an advantage originating from global training, ensemble architectures built from multiple graph-based models bring the most significant accuracy gains.

8.2 A Broader Perspective

Before we conclude this dissertation, we would like to look at the described contributions from a broader perspective. Specifically, two themes run throughout this thesis: understanding the behavior of statistical models and balancing between architectural complexity and accuracy. We discuss these two themes in more detail below.

8.2.1 Understanding the Behavior of Statistical Models

In Chapter 3, we provided examples of practical applications of ensemble parsers that can be found in the literature. We discussed such use cases as providing robust input for downstream tasks or supporting resource creation. However, one practical application of ensembles is commonly overlooked in the literature – their usability as *analysis tools*.

As a matter of fact, when Nivre and McDonald (2008) introduced feature-based stacking, they did not only demonstrate that this method had the potential of improving over the performance of individual models. They also confirmed that the theoretical strengths of transition- and graph-based models are indeed complementary in practice. In a similar matter, blending can be thought of as a method to examine diversity among the integrated models. For example, de Lhoneux et al. (2019) combined models that involved forward, backward, and bidirectional LSTMs to demonstrate that they learn different information.

In this context, this dissertation made contributions of two sorts: (1) we provided guidelines on practical applications of ensemble dependency parsers, but also (2) *through the ensembles*, we developed a deeper understanding of single parsing models. Since Section 8.1 already summarizes the contributions of this dissertation with respect to (1), we focus here on (2) on which we elaborate below.

Understanding dependency parsers. Ensemble methods commonly took up the function of analysis tools throughout this dissertation. Among others, oracle experiments demonstrated the upper bounds of transition- and graph-based paradigms. Ensemble methods provided several clues to understand the lack of diversity among BiLSTM-based parsers, which in turn triggered our investigations of the way these models encode structural information. Finally, through multi-task integration, we demonstrated the influence of local and global training on the representations trained with dependency parsers.

Clearly, although the goal of this thesis was to investigate ensemble architectures, through these ensembles, we have developed a deeper understanding of the behavior of

their individual building blocks.

Understanding neural architectures. Ensemble methods were not the only analysis tools that we employed. Throughout this dissertation, we also developed new ways of analyzing the behavior of parsers, for example, the cutting points discussed in Chapter 5 (Section 5.2). Especially important from this perspective was the IMPACT metric, which we used in Chapters 6 and 7 to analyze BiLSTM representations.

Recurrent Neural Network (RNN), which BiLSTMs are a variant of, has been repeatedly analyzed to understand whether it can learn syntactic relations. Such analyses differ in terms of (1) tasks on which the representations are trained on, and (2) the methodology they employ to probe what type of knowledge the representations have learned. Shi et al. (2016) demonstrated that sequence-to-sequence machine-translation systems capture source-language syntactic relations. Linzen et al. (2016) showed that when trained on the task of number agreement prediction, the representations capture a non-trivial amount of grammatical structure. Blevins et al. (2018) found that RNN representations trained on a variety of NLP tasks (including dependency parsing) can induce syntactic features (e.g., constituency labels of parent or grandparent) even without explicit supervision. Finally, Conneau et al. (2018) designed a set of tasks probing linguistic knowledge of sentence embedding methods.

From the perspective of the involved tasks, our work contributed to this line of research by demonstrating how BiLSTM-based dependency parsers take advantage of structural information encoded in the representations. However, it also contributed to the purely methodological perspective because, with the IMPACT metric, we showed how to employ derivatives to pinpoint what syntactic relations the representations are able to learn. Therefore, we believe that this method can contribute to a high-level NLP toolbox for investigating the behavior of neural models, not only dependency parsers.

8.2.2 Balancing between the Complexity and Accuracy of Architectures

The second theme that runs throughout this dissertation is the trade-off between the complexity of an architecture and its accuracy. Among others, we looked into ensembles that involved weaker but faster models. We searched for conditions under which monolingual models provide more robust results than large-scale cross-lingual techniques. Finally, we investigated sharing representations between transition- and graph-based models as a strategy to improve the performance of the former without additional processing cost at parsing time.

Methodological perspectives. A clear pattern emerges from our findings: in applications where the complexity of the architecture is not a bottleneck, a bulky architecture with strong single models is the most reliable approach. However, when the final accuracy of the system can be sacrificed, the decision on how to design the architecture becomes more challenging. Especially for such applications, *guidelines that take into account diverse methodological perspectives are needed*. In this dissertation, we focused on three such perspectives – parsing time, availability of training data, and parsing accuracy. However, there are other important questions that the end users have to face and that we leave open for future work. Answering questions such as "*Which ensemble approach is the best for the particular downstream task?*" or "*How to take into consideration the specificity of the language to be parsed?*" could extend the findings of this dissertation.

Ensemble dependency parsers. In Chapter 3 (Section 3.1.4), we noticed that ensemble methods have received less attention since the introduction of neural models, probably because neural parsers constantly improve the state of the art in the parsing field and, therefore, methods for improving their performance are currently less needed.

However, the question of *how to balance the complexity and the accuracy of the architectures* becomes even more important now than ever. The neural systems pay for their impressive performance with more complex architectures and the overhead of pre-training their representations. For example, deep contextualized BERT representations that we referred to in Chapter 6 take four days and 16 devices to be pre-trained and require 0.11 billion parameters (Devlin et al., 2019). In contrast, the most recent GPT-3 representations take weeks to train and use 175 billion parameters (Brown et al., 2020).

As a counterbalance, searching for greener and more efficient neural dependency parsers is recently gaining attention. Such approaches involve single light models, such as sequence labelers (Strzyz et al., 2019), but also ensemble methods. Interestingly, in the latter case, ensembles are used to obtain *less complex* solutions. For example, Anderson and Gómez-Rodríguez (2020) proposed an ensemble architecture with two dependency parsers, in which a larger model is compressed into a smaller one to increase its speed and decrease the number of parameters. Kuncoro et al. (2016) showed that multiple transition-based parsers could be distilled into a single graph-based parser, without significant sacrifice of the accuracy.

In this regard, we believe that the interest in ensemble architectures will grow and that they will serve as methods for combining light and weak models to produce robust results. Therefore, the work that has been presented in this dissertation can provide insights for designing new ensemble methods for neural dependency parsers. For exam-

ple, our preliminary experiments in this direction (Chapter 6, Section 6.4.2) showed the feasibility of multi-task learning within this context.

Appendix A

Treebank Statistics

A.1 SPMRL 2014 Shared Task Datasets

Language	Treebank references
Arabic	Maamouri et al. (2004); Habash and Roth (2009)
	Habash et al. (2009); Green and Manning (2010)
Basque	Aduriz et al. (2003)
French	Abeillé et al. (2003)
Hebrew	Sima'an et al. (2001); Tsarfaty (2010); Goldberg (2011)
	Tsarfaty (2013)
German	Brants et al. (2002); Seeker and Kuhn (2012)
Hungarian	Csendes et al. (2005); Vincze et al. (2010)
Korean	Choi et al. (1994); Choi (2013)
Polish	Świdziński and Woliński (2010)
Swedish	Nivre et al. (2006b)

Table A.1: References for SPMRL 2014 shared task treebanks (Seddah et al., 2014).

Language	Training set number of sentences	Development set number of sentences
Arabic	15762	1985
Basque	7577	948
French	14759	1235
Hebrew	5000	500
German	40472	5000
Hungarian	8146	1051
Korean	23010	2066
Polish	6578	821
Swedish	4999	493

Table A.2: Statistics for SPMRL 2014 shared task treebanks (Seddah et al., 2014).

A.2 Universal Dependencies ver. 2.0

Language	ISO code	Training set number of sent.	Development set number of sent.
Ancient_Greek	grc	11476	1137
Ancient_Greek-PROIEL	grc_proiel	14846	1019
Arabic	ar	6075	909
Basque	eu	5396	1798
Bulgarian	bg	8907	1115
Catalan	ca	13123	1709
Chinese	zh	3997	500
Croatian	hr	7689	600
Czech	cs	68495	9270
Czech-CAC	cs_cac	23478	603
Czech-CLTT	cs_cltt	465	349
Danish	da	4383	564
Dutch	nl	12330	720
Dutch-LassySmall	nl_lassysmall	6041	800
English	en	12543	2002
English-LinES	en_lines	2738	912
English-ParTUT	en_partut	1090	500

Language	ISO code	Training set number of sent.	Development set number of sent.
Estonian	et	2263	909
Finnish	fi	12217	1364
Finnish-FTB	fi_ftb	14981	1875
French	fr	14553	1478
French-Sequoia	fr_sequoia	2231	412
Galician	gl	2276	863
German	de	14118	799
Gothic	got	3387	985
Greek	el	1662	403
Hebrew	he	5241	484
Hindi	hi	13304	1659
Hungarian	hu	910	441
Indonesian	id	4477	559
Italian	it	12838	564
Japanese	ja	7164	511
Korean	ko	4400	950
Latin-ITTB	la_ittb	15808	700
Latin-PROIEL	la_proiel	14192	1132
Latvian	lv	2313	741
Norwegian-Bokmaal	no_bokmaal	15696	2410
Norwegian-Nynorsk	no_nynorsk	14174	1890
Old_Church_Slavonic	cu	4123	1073
Persian	fa	4798	599
Polish	pl	6100	1027
Portuguese	pt	8331	560
Portuguese-BR	pt_br	9664	1210
Romanian	ro	8043	752
Russian	ru	3850	579
Russian-SynTagRus	ru_syntagrus	48814	6584
Slovak	sk	8483	1060
Slovenian	sl	6478	734
Spanish	es	14187	1400
Spanish-AnCora	es_ancora	14305	1654

Language	ISO code	Training set number of sent.	Development set number of sent.
Swedish	sv	4303	504
Swedish-LinES	sv_lines	2738	912
Turkish	tr	3685	975
Urdu	ur	4043	552
Vietnamese	vi	1400	800

Table A.3: Statistics for the treebanks from Universal Dependencies v. 2.0 that have development sets (Nivre et al., 2017).

A.3 Universal Dependencies ver. 2.4

Language	ISO code	Training set number of sentences	Development set number of sentences
Arabic	ar_padt	6075	909
Basque	eu_bdt	5396	1798
Chinese	zh_gsd	3997	500
English	en_ewt	12543	2002
Finnish	fi_tdt	12217	1364
Hebrew	he_htb	5241	484
Hindi	hi_hdtb	13304	1659
Italian	it_isdt	13121	564
Japanese	ja_gsd	7125	511
Korean	ko_gsd	4400	950
Russian	ru_syntagrus	48814	6584
Swedish	sv_talbanken	4303	504
Turkish	tr_imst	3664	988

Table A.4: A selection of treebanks from Universal Dependencies v. 2.4 (Nivre et al., 2019) used for experiments in Chapter 6.

Appendix B

Hyperparameters and Additional Analysis

B.1 Appendix for Chapter 2

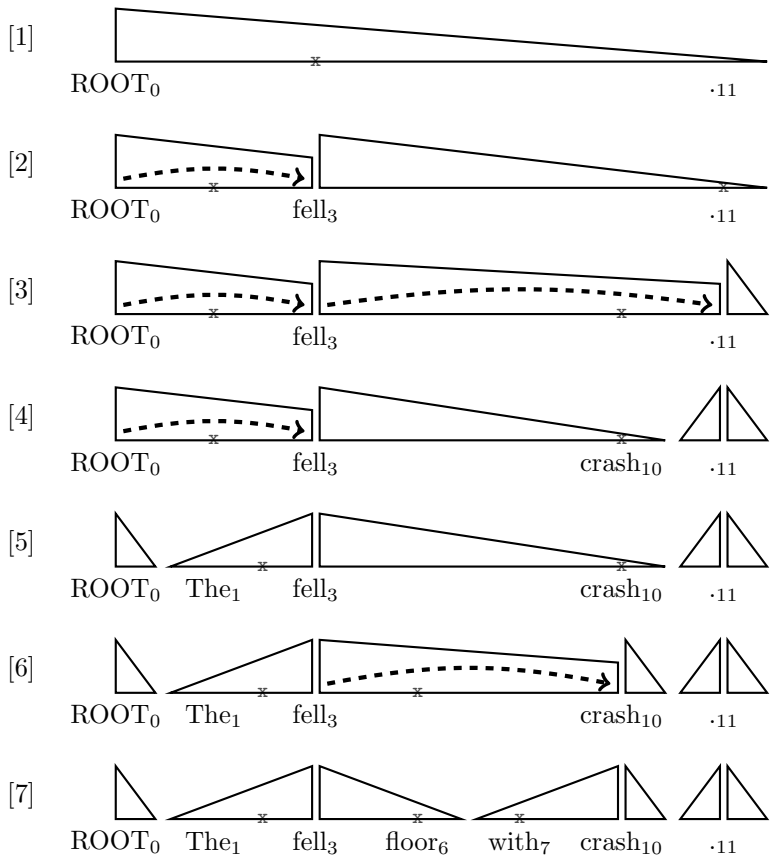


Figure B.1: Example of backtracking in Eisner's algorithm. First seven steps to get the tree from Figure 1.1. Back-pointers are marked with small x.

Figure B.1 shows an example of the first seven steps of the backtracking procedure applied to the sentence from Figure 1.1. It starts from the left-closed structure that spans the whole sentence. The structure also keeps information that it was built from an open structure spanning between 0 and 3 and a closed one starting at 3 (marked with x on the figure). This means that the final tree will contain an arc $\text{ROOT}_0 \rightarrow \textit{fell}_3$ (see step [2]). The procedure continues and, in the first seven steps, finds two more arcs: $\textit{fell}_3 \rightarrow .{}_{11}$ in step [3] and $\textit{fell}_3 \rightarrow \textit{crash}_{10}$ in step [6].

B.2 Appendix for Chapter 6

Word embedding dimension	300
POS tag embedding dimension	20
Character embedding dimension	24
Supertag embedding dimension	30
ELMo representation dimension	1024
Hidden units in MLP	100
LSTM layers	2
LSTM dimensions	125
LSTM dropout	0.33
Character-based LSTM dimensions	100
α for word dropout	0.25
Trainer	Adam
Non-lin function	tanh

Table B.1: Hyperparameters for the parsers used in experimentation in Chapter 6.

	TB	$\text{STACK}_{\text{GB}}^{\text{TB}}$	GB	$\text{STACK}_{\text{TB}}^{\text{GB}}$
Arabic	0.089	0.145	0.159	0.131
Basque	0.297	0.212	0.314	0.234
Chinese	0.495	0.200	0.261	0.351
English	0.184	0.180	0.166	0.143
Finnish	0.162	0.176	0.216	0.105
Hebrew	0.250	0.136	0.226	0.339
Hindi	0.169	0.093	0.130	0.116
Italian	0.201	0.136	0.159	0.067
Japanese	0.213	0.158	0.108	0.105
Korean	0.432	0.288	0.414	0.234
Russian	0.116	0.042	0.064	0.056
Swedish	0.153	0.209	0.174	0.171
Turkish	0.571	0.406	0.280	0.459

Table B.2: Standard deviation for the results in Table 6.1.

	TB	$\text{MTL}_{\text{GB}}^{\text{TB}}$	GB	$\text{MTL}_{\text{TB}}^{\text{GB}}$
Arabic	0.089	0.249	0.159	0.225
Basque	0.297	0.325	0.314	0.170
Chinese	0.495	0.604	0.261	0.425
English	0.184	0.144	0.166	0.058
Finnish	0.162	0.385	0.216	0.237
Hebrew	0.250	0.368	0.226	0.356
Hindi	0.169	0.134	0.130	0.081
Italian	0.201	0.409	0.159	0.298
Japanese	0.213	0.167	0.108	0.157
Korean	0.432	0.239	0.414	0.319
Russian	0.116	0.078	0.064	0.087
Swedish	0.153	0.167	0.174	0.152
Turkish	0.571	0.336	0.280	0.290

Table B.3: Standard deviation for the results in Table 6.3.

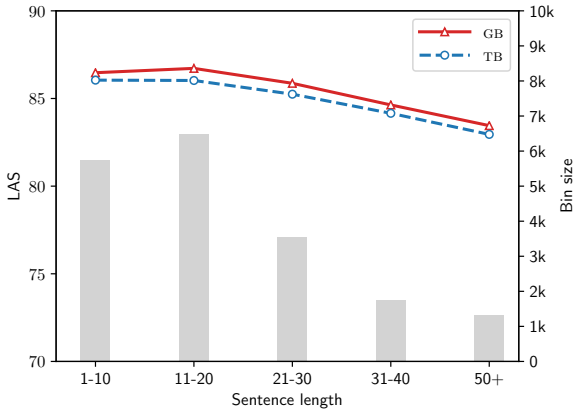
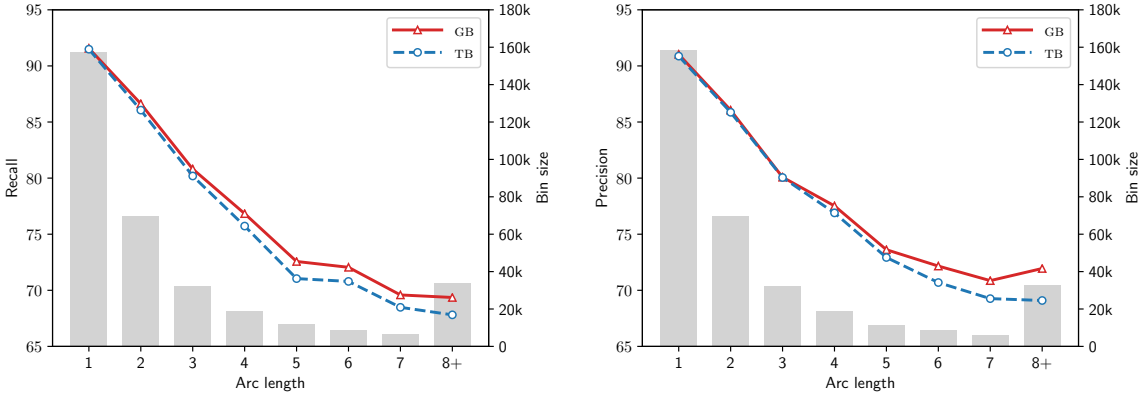
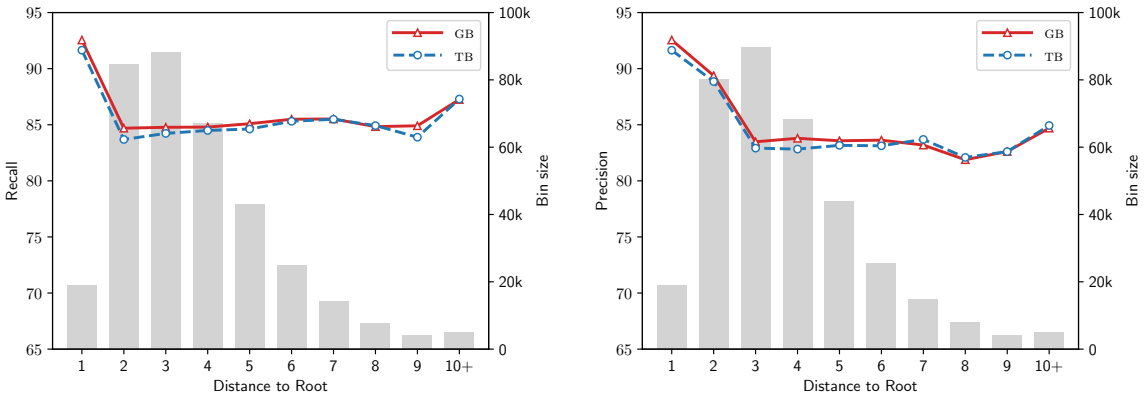


Figure B.2: Average LAS relative to sentence length on development sets (experimental setting described in Chapter 6).



(a) Recall **(b)** Precision

Figure B.3: The dependency recall and precision relative to arc length on development sets (experimental setting described in Chapter 6).



(a) Recall **(b)** Precision

Figure B.4: The dependency recall and precision relative to distance to root on development sets (experimental setting described in Chapter 6).

B.3 Appendix for Chapter 7

Word embedding dimension	100
POS tag embedding dimension	20
Hidden units in MLP	100
LSTM layers	2
LSTM dimensions	125
α for word dropout	0.25
Trainer	Adam
Non-lin function	tanh

Table B.4: Hyperparameters for the parsers used in experiments in Chapter 7.

	TbMin	TbExt	GbMin	GbSibl
Arabic	0.323	0.191	0.179	0.186
Chinese	0.398	0.267	0.408	0.441
English	0.207	0.176	0.212	0.149
English PTB	0.237	0.211	0.146	0.103
Finnish	0.163	0.323	0.157	0.219
Greek	0.382	0.472	0.340	0.372
Hebrew	0.391	0.454	0.269	0.229
Korean	0.740	0.456	0.300	0.163
Russian	0.282	0.408	0.228	0.169
Swedish	0.295	0.257	0.379	0.195

Table B.5: Standard deviation for the results in Table 7.1.

Bibliography

Abeillé, A., Clément, L., and Toussenel, F. (2003). Building a Treebank for French. In Abeillé, A., editor, *Treebanks*, pages 165–187. Kluwer, Dordrecht. Cited on page 165.

Aduriz, I., Aranzabe, M. J., Arriola, J. M., Atutxa, A., Díaz de Ilarraza, A., Garmendia, A., and Oronoz, M. (2003). Construction of a Basque Dependency Treebank. In *TLT-03*, pages 201–204. Cited on page 165.

Agić, Ž. (2017). Cross-Lingual Parser Selection for Low-Resource Languages. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*, pages 1–10, Gothenburg, Sweden. Association for Computational Linguistics. Cited on pages 5, 106, 107, and 108.

Agić, Ž., Johannsen, A., Plank, B., Martínez Alonso, H., Schluter, N., and Søgaard, A. (2016). Multilingual Projection for Parsing Truly Low-Resource Languages. *Transactions of the Association for Computational Linguistics*, 4:301–312. Cited on pages 103 and 118.

Ahrenberg, L. (2007). LinES: An English-Swedish Parallel Treebank. In *Proceedings of the 16th Nordic Conference of Computational Linguistics (NODALIDA 2007)*, pages 270–273, Tartu, Estonia. University of Tartu, Estonia. Cited on page 15.

Akbik, A., Chiticariu, L., Danilevsky, M., Li, Y., Vaithyanathan, S., and Zhu, H. (2015). Generating High Quality Proposition Banks for Multilingual Semantic Role Labeling. 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)*, pages 397–407, Beijing, China. Association for Computational Linguistics. Cited on page 18.

Ambati, B. R., Deoskar, T., and Steedman, M. (2014). Improving Dependency Parsers using Combinatory Categorial Grammar. In *Proceedings of the 14th Conference of the*

European Chapter of the Association for Computational Linguistics, volume 2: Short Papers, pages 159–163, Gothenburg, Sweden. Association for Computational Linguistics. Cited on pages 80, 88, 96, and 99.

Anderson, M. and Gómez-Rodríguez, C. (2020). Distilling Neural Networks for Greener and Faster Dependency Parsing. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 2–13, Online. Association for Computational Linguistics. Cited on page 162.

Attardi, G. (2006). Experiments with a Multilanguage Non-Projective Dependency Parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 166–170, New York City. Association for Computational Linguistics. Cited on pages 41 and 121.

Attardi, G. and Dell'Orletta, F. (2009). Reverse Revision and Linear Tree Combination for Dependency Parsing. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 261–264, Boulder, Colorado. Association for Computational Linguistics. Cited on pages 64, 65, 77, 79, and 80.

Attardi, G., Dell'Orletta, F., Simi, M., and Turian, J. (2009). Accurate Dependency Parsing with a Stacked Multilayer Perceptron. In *Proceeding of Evalita 2009*, LNCS. Springer. Cited on pages 42 and 44.

Aufrant, L., Wisniewski, G., and Yvon, F. (2016). Zero-resource Dependency Parsing: Boosting Delexicalized Cross-lingual Transfer with Linguistic Knowledge. In *COLING*, pages 119–130. Cited on page 102.

Baker, C. F., Fillmore, C. J., and Lowe, J. B. (1998). The Berkeley FrameNet Project. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1*, pages 86–90, Montreal, Quebec, Canada. Association for Computational Linguistics. Cited on page 22.

Ballesteros, M., Dyer, C., and Smith, N. A. (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*, pages 349–359, Lisbon, Portugal. Association for Computational Linguistics. Cited on page 52.

Bangalore, S. and Joshi, A. K. (1999). Supertagging: An Approach to Almost Parsing. *Computational Linguistics*, 25(2):237–265. Cited on page 80.

Bender, E. M. (2011). On Achieving and Evaluating Language-Independence in NLP. *Linguistic Issues in Language Technology*, 6(3):1–26. Cited on page 102.

Bies, A., Mott, J., Warner, C., and Kulick, S. (2012). English Web Treebank LDC2012T13. Cited on pages 82 and 96.

Bjerva, J. (2017). *One Model to Rule them All: multitask and Multilingual Modelling for Lexical Analysis*. PhD thesis, University of Groningen. Cited on page 48.

Björkelund, A., Çetinoğlu, Ö., Faleńska, A., Farkas, R., Mueller, T., Seeker, W., and Szántó, Z. (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*, pages 97–102, Dublin, Ireland. Dublin City University. Cited on page 84.

Björkelund, A., Cetinoglu, O., Farkas, R., Mueller, T., and Seeker, W. (2013). (Re)ranking Meets Morphosyntax: State-of-the-art Results from the SPMRL 2013 Shared Task. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 135–145, Seattle, Washington, USA. Association for Computational Linguistics. Cited on pages 69 and 83.

Björkelund, A., Faleńska, A., Seeker, W., and Kuhn, J. (2016). How to Train Dependency Parsers with Inexact Search for Joint Sentence Boundary Detection and Parsing of Entire Documents. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1924–1934, Berlin, Germany. Association for Computational Linguistics. Cited on page 20.

Björkelund, A., Falenska, A., Yu, X., and Kuhn, J. (2017). IMS at the CoNLL 2017 UD Shared Task: CRFs and Perceptrons Meet Neural Networks. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 40–51, Vancouver, Canada. Association for Computational Linguistics. Cited on pages 20, 69, and 83.

Björkelund, A. and Nivre, J. (2015). Non-Deterministic Oracles for Unrestricted Non-Projective Transition-Based Dependency Parsing. In *Proceedings of the 14th International*

Conference on Parsing Technologies, pages 76–86, Bilbao, Spain. Association for Computational Linguistics. Cited on page 85.

Blevins, T., Levy, O., and Zettlemoyer, L. (2018). Deep RNNs Encode Soft Hierarchical Syntax. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 14–19, Melbourne, Australia. Association for Computational Linguistics. Cited on page 161.

Blum, A. and Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100. Cited on page 66.

Bohnet, B. (2010). Top Accuracy and Fast Dependency Parsing is not a Contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 89–97, Beijing, China. Coling 2010 Organizing Committee. Cited on pages 83 and 111.

Bohnet, B. and Kuhn, J. (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*, pages 77–87, Avignon, France. Association for Computational Linguistics. Cited on page 66.

Bohnet, B. and Nivre, J. (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*, pages 1455–1465, Jeju Island, Korea. Association for Computational Linguistics. Cited on pages 20, 83, and 111.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 144–152, New York, NY, USA. Association for Computing Machinery. Cited on page 41.

Bottou, L. (2012). Stochastic Gradient Descent Tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, Berlin, Heidelberg. Cited on page 55.

Bouma, G., Seddah, D., and Zeman, D. (2020). Overview of the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced*

Universal Dependencies, pages 151–161, Online. Association for Computational Linguistics. Cited on page 69.

Brants, S., Dipper, S., Eisenberg, P., Hansen-Schirra, S., König, E., Lezius, W., Rohrer, C., Smith, G., and Uszkoreit, H. (2004). TIGER: Linguistic Interpretation of a German Corpus. *Research on Language and Computation*, 2(4):597–620. Cited on page 17.

Brants, S., Dipper, S., Hansen, S., Lezius, W., and Smith, G. (2002). The TIGER Treebank. In Hinrichs, E. and Simov, K., editors, *Proceedings of the First Workshop on Treebanks and Linguistic Theories (TLT 2002)*, pages 24–41, Sozopol, Bulgaria. Cited on page 165.

Breiman, L. (1996). Stacked regressions. *Machine Learning*, 24(1):49–64. Cited on page 65.

Brill, E. and Wu, J. (1998). Classifier Combination for Improved Lexical Disambiguation. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1*, pages 191–195, Montreal, Quebec, Canada. Association for Computational Linguistics. Cited on page 60.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language Models are Few-Shot Learners. *arXiv preprint arXiv:2005.14165*. Cited on page 162.

Buchholz, S. and Marsi, E. (2006). CoNLL-X Shared Task on Multilingual Dependency Parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, New York City. Association for Computational Linguistics. Cited on pages 4 and 68.

Buckwalter, T. (2002). Buckwalter Arabic Morphological Analyzer Version 1.0. *Linguistic Data Consortium, University of Pennsylvania, 2002. LDC Catalog No.: LDC2002L49.* Cited on page 84.

Carreras, X. (2007). Experiments with a Higher-Order Projective Dependency Parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 957–961, Prague, Czech Republic. Association for Computational Linguistics. Cited on pages 40 and 111.

Caruana, R. (1993). Multitask Learning: A Knowledge-Based Source of Inductive Bias. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 41–48. Morgan Kaufmann. Cited on page 123.

Charniak, E. and Johnson, M. (2005). Coarse-to-Fine n-Best Parsing and MaxEnt Discriminative Reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 173–180, Ann Arbor, Michigan. Association for Computational Linguistics. Cited on page 67.

Che, W., Guo, J., Wang, Y., Zheng, B., Zhao, H., Liu, Y., Teng, D., and Liu, T. (2017). The HIT-SCIR System for End-to-End Parsing of Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 52–62, Vancouver, Canada. Association for Computational Linguistics. Cited on page 68.

Che, W., Liu, Y., Wang, Y., Zheng, B., and Liu, T. (2018). Towards Better UD Parsing: Deep Contextualized Word Embeddings, Ensemble, and Treebank Concatenation. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 55–64, Brussels, Belgium. Association for Computational Linguistics. Cited on pages 51, 52, 68, 69, 121, 122, 123, and 124.

Chen, D. and Manning, C. (2014). A Fast and Accurate Dependency Parser using Neural Networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750. Association for Computational Linguistics. Cited on pages ix, 42, 43, 44, 45, 46, 47, 49, 50, 121, 125, and 146.

Choi, J. D. (2013). Preparing Korean Data for the Shared Task on Parsing Morphologically Rich Languages. *arXiv preprint arXiv:1309.1649*. Cited on page 165.

Choi, J. D., Tetreault, J., and Stent, A. (2015). It Depends: Dependency Parser Comparison Using A Web-based Evaluation Tool. 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)*, pages 387–396, Beijing, China. Association for Computational Linguistics. Cited on pages 4 and 60.

Choi, K.-S., Han, Y. S., Han, Y. G., and Kwon, O. W. (1994). KAIST Tree Bank Project for Korean: Present and Future Development. In *Proceedings of the International Workshop on Sharable Natural Language Resources*, pages 7–14. Citeseer. Cited on page 165.

Christodouloupoulos, C. and Steedman, M. (2015). A massively parallel corpus: the Bible in 100 languages. *Language Resources and Evaluation*, 49(2):375. Cited on page 103.

Chu, Y. and Liu, T. (1965). On the shortest aborescence of a directed graph. *Science Sinica*, 14:1396–1400. Cited on pages 34, 125, and 143.

Clark, S., Copestake, A., Curran, J. R., Zhang, Y., Herbelot, A., Haggerty, J., Ahn, B.-G., Wyk, C. V., Roesner, J., Kummerfeld, J., and Dawborn, T. (2009). Large-Scale Syntactic Processing : Parsing the Web. In *Final Report of the 2009 JHU CLSP Workshop*. Cited on page 5.

Clark, S. and Curran, J. R. (2004). The Importance of Supertagging for Wide-Coverage CCG Parsing. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 282–288, Geneva, Switzerland. COLING. Cited on page 80.

Cocke, J. (1969). *Programming languages and their compilers: Preliminary notes*. New York University. Cited on page 38.

Collins, M. (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 (EMNLP 2002)*, pages 1–8. Association for Computational Linguistics. Cited on pages 54 and 55.

Collins, M. and Koo, T. (2005). Discriminative Reranking for Natural Language Parsing. *Computational Linguistics*, 31(1):25–70. Cited on page 67.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493–2537. Cited on page 44.

Conneau, A., Kruszewski, G., Lample, G., Barrault, L., and Baroni, M. (2018). What you can cram into a single $&!#* vector: Probing sentence embeddings for linguistic properties. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2126–2136, Melbourne, Australia. Association for Computational Linguistics. Cited on page 161.

Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., and Singer, Y. (2006). Online Passive–Aggressive Algorithms. *Journal of Machine Learning Reseach*, 7:551–585. Cited on pages 54, 85, and 111.

Cross, J. and Huang, L. (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)*, pages 32–37. Association for Computational Linguistics. Cited on pages 49 and 52.

Csendes, D., Csirik, J., Gyimóthy, T., and Kocsor, A. (2005). The Szeged Treebank. In Matoušek, V., Mautner, P., and Pavelka, T., editors, *International Conference on Text, Speech and Dialogue*, pages 123–131. Springer. Cited on page 165.

Dakota, D. and Kübler, S. (2018). Practical Parsing for Downstream Applications. In *Proceedings of the 27th International Conference on Computational Linguistics: Tutorial Abstracts*, pages 5–7, Santa Fe, New Mexico, USA. Association for Computational Linguistics. Cited on page 4.

Daumé, H. (2015). *A Course In Machine Learning*. Self Published. Cited on page 53.

de Lhoneux, M., Ballesteros, M., and Nivre, J. (2019). Recursive Subtree Composition in LSTM-Based Dependency Parsing. 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)*, pages 1566–1576, Minneapolis, Minnesota. Association for Computational Linguistics. Cited on pages 134, 137, and 160.

de Lhoneux, M., Shao, Y., Basirat, A., Kiperwasser, E., Stymne, S., Goldberg, Y., and Nivre, J. (2017). From Raw Text to Universal Dependencies - Look, No Tags! In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 207–217. Association for Computational Linguistics. Cited on page 52.

de Marneffe, M.-C., Dozat, T., Silveira, N., Haverinen, K., Ginter, F., Nivre, J., and Manning, C. D. (2014). Universal Stanford dependencies: A cross-linguistic typology. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 4585–4592, Reykjavik, Iceland. European Language Resources Association (ELRA). Cited on pages 17 and 18.

De Marneffe, M.-C. and Nivre, J. (2019). Dependency grammar. *Annual Review of Linguistics*, 5:197–218. Cited on pages 14 and 19.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (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)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics. Cited on pages 52, 121, 123, and 162.

Dietterich, T. G. (2000). Ensemble Methods in Machine Learning. In *International Workshop on Multiple Classifier Systems*, pages 1–15, Berlin, Heidelberg. Springer. Cited on pages 3, 60, and 76.

Domeij, R., Knutsson, O., Carlberger, J., and Kann, V. (2000). Granska–an efficient hybrid system for Swedish grammar checking. In *Proceedings of the 12th Nordic Conference of Computational Linguistics (NODALIDA 1999)*, pages 49–56. Cited on page 84.

Dozat, T. and Manning, C. D. (2017). Deep Biaffine Attention for Neural Dependency Parsing. In *Proceedings of the 5th International Conference on Learning Representations*, ICLR'17. Cited on pages 50, 52, and 123.

Dozat, T., Qi, P., and Manning, C. D. (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*, pages 20–30, Vancouver, Canada. Association for Computational Linguistics. Cited on pages 51, 69, and 122.

Dryer, M. S. and Haspelmath, M., editors (2013). *WALS Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig. Cited on pages 102 and 107.

Duong, L., Cohn, T., Bird, S., and Cook, P. (2015). A Neural Network Model for Low-Resource Universal Dependency Parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 339–348, Lisbon, Portugal. Association for Computational Linguistics. Cited on page 104.

Dyer, C., Ballesteros, M., Ling, W., Matthews, A., and Smith, N. A. (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)*, pages 334–343. Association for Computational Linguistics. Cited on page 49.

Echelmeyer, N., Reiter, N., and Schulz, S. (2017). Ein PoS-Tagger für „das" Mittelhochdeutsche. In *Book of Abstracts of DHd 2017*, pages 141–147, Bern, Switzerland. Cited on page 105.

Eckart, K. (2018). *Task-based parser output combination: workflow and infrastructure*. Ph.D. dissertation, Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart, Stuttgart. Cited on page 73.

Edmonds, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71(B):233–240. Cited on pages 34, 125, and 143.

Eisner, J. M. (1996). Three New Probabilistic Models for Dependency Parsing: An Exploration. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*. Cited on pages 30, 38, 40, 111, 124, and 143.

Elman, J. L. (1990). Finding Structure in Time. *Cognitive science*, 14(2):179–211. Cited on page 47.

Faleńska, A., Björkelund, A., Çetinoğlu, Ö., and Seeker, W. (2015). Stacking or Supertagging for Dependency Parsing – What's the Difference? In *Proceedings of the 14th International Conference on Parsing Technologies*, pages 118–129, Bilbao, Spain. Association for Computational Linguistics. Cited on page 79.

Falenska, A., Björkelund, A., and Kuhn, J. (2020a). Integrating Graph-Based and Transition-Based Dependency Parsers in the Deep Contextualized Era. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 25–39, Online. Association for Computational Linguistics. Cited on page 121.

Falenska, A., Björkelund, A., Yu, X., and Kuhn, J. (2018). IMS at the PolEval 2018: A Bulky Ensemble Dependency Parser meets 12 Simple Rules for Predicting Enhanced Dependencies in Polish. In Ogrodniczuk, M. and Łukasz Kobyliński, editors, *Proceedings of the PolEval 2018 Workshop*, pages 25–39, Warsaw, Poland. Institute of Computer Science, Polish Academy of Sciences. Cited on pages 71 and 72.

Falenska, A. and Çetinoğlu, Ö. (2017). Lexicalized vs. Delexicalized Parsing in Low-Resource Scenarios. In *Proceedings of the 15th International Conference on Parsing Technologies*, pages 18–24, Pisa, Italy. Association for Computational Linguistics. Cited on page 101.

Falenska, A., Czesznak, Z., Jung, K., Völkel, M., Seeker, W., and Kuhn, J. (2020b). GRAIN-S: Manually Annotated Syntax for German Interviews. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 5169–5177, Marseille, France. European Language Resources Association. Cited on page 75.

Falenska, A. and Kuhn, J. (2019). The (Non-)Utility of Structural Features in BiLSTM-based Dependency Parsers. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 117–128, Florence, Italy. Association for Computational Linguistics. Cited on page 139.

Farkas, R. and Bohnet, B. (2012). Stacking of Dependency and Phrase Structure Parsers. In *Proceedings of COLING 2012*, pages 849–866, Mumbai, India. The COLING 2012 Organizing Committee. Cited on page 66.

Fernandez-Gonzalez, D., Gomez-Rodriguez, C., and Vilares, D. (2016). Improving the Arc-Eager Model with Reverse Parsing. *Computing and Informatics*, 35(3):555–585. Cited on page 64.

Fishel, M. and Nivre, J. (2009). Voting and Stacking in Data-Driven Dependency Parsing. In *Proceedings of 17th Nordic Conference on Computational Linguistics (NODALIDA 2009)*, pages 219–222, Odense, Denmark. Cited on page 64.

Flannery, D. and Mori, S. (2015). Combining Active Learning and Partial Annotation for Domain Adaptation of a Japanese Dependency Parser. In *Proceedings of the 14th International Conference on Parsing Technologies*, pages 11–19. Cited on pages 74 and 75.

Florian, R. and Yarowsky, D. (2002). Modeling Consensus: Classifier Combination for Word Sense Disambiguation. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pages 25–32. Association for Computational Linguistics. Cited on page 60.

Forcada, M. L., Ginestí-Rosell, M., Nordfalk, J., O'Regan, J., Ortiz-Rojas, S., Pérez-Ortiz, J. A., Sánchez-Martínez, F., Ramírez-Sánchez, G., and Tyers, F. M. (2011). Apertium: a free/open-source platform for rule-based machine translation. *Machine translation*, 25(2):127–144. Cited on page 84.

Foth, K. A., By, T., and Menzel, W. (2006). Guiding a Constraint Dependency Parser with Supertags. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 289–296, Sydney, Australia. Association for Computational Linguistics. Cited on pages 80 and 83.

Foth, K. A., Daum, M., and Menzel, W. (2004). Interactive grammar development with WCDG. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, pages 122–125, Barcelona, Spain. Association for Computational Linguistics. Cited on page 80.

Freund, Y. and Schapire, R. E. (1999). Large Margin Classification Using the Perceptron Algorithm. *Machine learning*, 37(3):277–296. Cited on page 55.

Gaddy, D., Stern, M., and Klein, D. (2018). What's Going On in Neural Constituency Parsers? An Analysis. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 999–1010. Association for Computational Linguistics. Cited on pages 133, 134, and 140.

Garrette, D. and Baldridge, J. (2013). Learning a Part-of-Speech Tagger from Two Hours of Annotation. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 138–147, Atlanta, Georgia. Association for Computational Linguistics. Cited on pages 73 and 105.

Goldberg, Y. (2011). *Automatic Syntactic Processing of Modern Hebrew*. PhD thesis, Ben Gurion University of the Negev. Cited on page 165.

Goldberg, Y. (2017). *Neural Network Methods for Natural Language Processing*, volume 10. Morgan & Claypool Publishers. Cited on pages 1, 41, and 43.

Goldberg, Y. and Elhadad, M. (2010). An Efficient Algorithm for Easy-First Non-Directional Dependency Parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750, Los Angeles, California. Association for Computational Linguistics. Cited on page 23.

Goldberg, Y. and Elhadad, M. (2013). Word Segmentation, Unknown-word Resolution, and Morphological Agreement in a Hebrew Parsing System. *Computational Linguistics*, 39(1):121–160. Cited on page 84.

Goldberg, Y. and Nivre, J. (2012). A Dynamic Oracle for Arc-Eager Dependency Parsing. In *Proceedings of COLING 2012*, pages 959–976, Mumbai, India. The COLING 2012 Organizing Committee. Cited on pages 25 and 124.

Goldberg, Y. and Nivre, J. (2013). Training Deterministic Parsers with Non-Deterministic Oracles. *Transactions of the Association for Computational Linguistics*, 1:403–414. Cited on pages 25 and 124.

Gómez-Rodríguez, C., Alonso-Alonso, I., and Vilares, D. (2017). How important is syntactic parsing accuracy? An empirical evaluation on rule-based sentiment analysis. *Artificial Intelligence Review*, pages 1–17. Cited on pages 5, 73, and 138.

Gómez-Rodríguez, C., Shi, T., and Lee, L. (2018). Global Transition-based Non-projective Dependency Parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2664–2675. Association for Computational Linguistics. Cited on page 155.

Grave, E., Bojanowski, P., Gupta, P., Joulin, A., and Mikolov, T. (2018). Learning Word Vectors for 157 Languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA). Cited on pages 124 and 141.

Graves, A. and Schmidhuber, J. (2005). Framewise Phoneme Classification with Bidirectional LSTM and Other Neural Network Architectures. *Neural Networks*, 18(5):602–610. Cited on pages 50 and 121.

Green, N. D. and Žabokrtský, Z. (2016). *Creating Hybrid Dependency Parsers for Syntax-Based MT*, pages 161–190. Springer International Publishing, Cham. Cited on pages 2 and 72.

Green, S. and Manning, C. D. (2010). Better Arabic Parsing: Baselines, Evaluations, and Analysis. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 394–402, Beijing, China. Coling 2010 Organizing Committee. Cited on page 165.

Guo, J., Che, W., Yarowsky, D., Wang, H., and Liu, T. (2015). Cross-lingual Dependency Parsing Based on Distributed Representations. In *Proceedings of ACL-IJCNLP*, pages 1234–1244, Beijing, China. Cited on pages 104 and 119.

Habash, N., Faraj, R., and Roth, R. (2009). Syntactic Annotation in the Columbia Arabic Treebank. In *Proceedings of MEDAR International Conference on Arabic Language Resources and Tools*, Cairo, Egypt. Cited on page 165.

Habash, N. and Roth, R. (2009). CATiB: The Columbia Arabic Treebank. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 221–224, Suntec, Singapore. Association for Computational Linguistics. Cited on page 165.

Hajič, J., Ciaramita, M., Johansson, R., Kawahara, D., Martí, M. A., Màrquez, L., Meyers, A., Nivre, J., Padó, S., Štěpánek, J., Straňák, P., Surdeanu, M., Xue, N., and Zhang, Y. (2009). The CoNLL-2009 Shared Task: Syntactic and Semantic Dependencies in Multiple Languages. In *Proceedings of the Thirteenth Conference on Computational Natural Lan-*

guage Learning (CoNLL 2009): Shared Task, pages 1–18, Boulder, Colorado. Association for Computational Linguistics. Cited on page 68.

Hajič, J., Hajičová, E., Pajas, P., Panevová, J., and Sgall, P. (2001). Prague Dependency Treebank, version 1.0 LDC2001T10. Cited on page 16.

Hall, J., Nilsson, J., Nivre, J., Eryigit, G., Megyesi, B., Nilsson, M., and Saers, M. (2007). Single Malt or Blended? A Study in Multilingual Parser Optimization. In *Proceedings of the CoNLL Shared Task Session of EMNLP-C oNLL 2007*, pages 933–939, Prague, Czech Republic. Association for Computational Linguistics. Cited on pages 63, 65, and 69.

Hall, K. (2007). K-best Spanning Tree Parsing. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 392–399, Prague, Czech Republic. Association for Computational Linguistics. Cited on page 67.

Havelka, J. (2007). Beyond Projectivity: Multilingual Evaluation of Constraints and Measures on Non-Projective Structures. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 608–615, Prague, Czech Republic. Association for Computational Linguistics. Cited on pages 15 and 16.

Hayashi, K., Kondo, S., and Matsumoto, Y. (2013). Efficient Stacked Dependency Parsing by Forest Reranking. *Transactions of the Association for Computational Linguistics*, 1:139–150. Cited on page 67.

Henderson, J. C. and Brill, E. (1999). Exploiting Diversity in Natural Language Processing: Combining Parsers. In *1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*. Cited on page 67.

Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780. Cited on pages 48 and 121.

Huang, B. and Carley, K. (2019). Syntax-Aware Aspect Level Sentiment Classification with Graph Attention Networks. 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)*, pages 5469–5477, Hong Kong, China. Association for Computational Linguistics. Cited on pages 2 and 6.

Hudson, R. A. (1984). *Word Grammar*. Basil Blackwell, Oxford, UK. Cited on page 13.

Hwa, R. (2004). Sample Selection for Statistical Parsing. *Computational Linguistics*, 30(3):253–276. Cited on pages 74 and 75.

Hwa, R., Resnik, P., Weinberg, A., Cabezas, C., and Kolak, O. (2005). Bootstrapping parsers via syntactic projection across parallel texts. *Natural language engineering*, 11(3):311–325. Cited on page 102.

Inui, T. and Inui, K. (2000). Committee-based Decision Making in Probabilistic Partial Parsing. In *COLING 2000 Volume 1: The 18th International Conference on Computational Linguistics*. Cited on page 60.

Jawahar, G., Muller, B., Fethi, A., Martin, L., Villemonte de la Clergerie, É., Sagot, B., and Seddah, D. (2018). ELMoLex: Connecting ELMo and Lexicon Features for Dependency Parsing. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 223–237, Brussels, Belgium. Association for Computational Linguistics. Cited on page 123.

Joshi, A. K. and Bangalore, S. (1994). Disambiguation of Super Parts of Speech (or Supertags): Almost Parsing. In *Proceedings of the 15th Conference on Computational Linguistics - Volume 1*, COLING '94, pages 154–160, Stroudsburg, PA, USA. Association for Computational Linguistics. Cited on pages 5, 80, and 81.

Jurafsky, D. and Martin, J. H. (2014). *Speech and Language Processing*, volume 3. Prentice Hall PTR. Cited on page 21.

Kanerva, J., Pyysalo, S., and Ginter, F. (2017). Fully Delexicalized Contexts for Syntax-Based Word Embeddings. In *Proceedings of the Fourth International Conference on Dependency Linguistics (Depling 2017)*, pages 83–91, Pisa,Italy. Linköping University Electronic Press. Cited on page 47.

Kasami, T. (1966). An Efficient Recognition and Syntax-Analysis Algorithm for Context-Free Languages. *Coordinated Science Laboratory Report no. R-257*. Cited on page 38.

Katz-Brown, J., Petrov, S., McDonald, R., Och, F. J., Talbot, D., Ichikawa, H., Seno, M., and Kazawa, H. (2011). Training a Parser for Machine Translation Reordering. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 183–192. Cited on page 73.

Kingma, D. P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*. Cited on page 55.

Kiperwasser, E. and Goldberg, Y. (2016a). Easy-First Dependency Parsing with Hierarchical Tree LSTMs. *Transactions of the Association for Computational Linguistics*, 4:445–461. Cited on page 49.

Kiperwasser, E. and Goldberg, Y. (2016b). Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations. *Transactions of the Association for Computational Linguistics*, 4:313–327. Cited on pages x, 6, 49, 50, 51, 52, 55, 57, 121, 122, 123, 125, 138, 139, 140, 141, 142, and 144.

Klein, D. and Manning, C. (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 (ACL-04)*, pages 478–485, Barcelona, Spain. Cited on page 3.

Kondratyuk, D. and Straka, M. (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)*, pages 2779–2795, Hong Kong, China. Association for Computational Linguistics. Cited on page 123.

Koo, T. and Collins, M. (2010). Efficient Third-Order Dependency Parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11, Uppsala, Sweden. Association for Computational Linguistics. Cited on page 40.

Koo, T., Rush, A. M., Collins, M., Jaakkola, T., and Sontag, D. (2010). Dual Decomposition for Parsing with Non-Projective Head Automata. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1288–1298, Cambridge, MA. Association for Computational Linguistics. Cited on page 40.

Kübler, S., McDonald, R., and Nivre, J. (2009). Dependency Parsing. *Synthesis Lectures on Human Language Technologies*, 1(1):1–127. Cited on pages 14, 21, 34, and 37.

Kuhlmann, M., Gómez-Rodríguez, C., and Satta, G. (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*, pages 673–682, Portland, Oregon, USA. Association for Computational Linguistics. Cited on pages 23 and 124.

Kuhlmann, M. and Nivre, J. (2006). Mildly Non-Projective Dependency Structures. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 507–514, Sydney, Australia. Association for Computational Linguistics. Cited on pages 15 and 16.

Kullback, S. and Leibler, R. A. (1951). On Information and Sufficiency. *The Annals of Mathematical Statistics*, pages 79–86. Cited on page 107.

Kulmizev, A., de Lhoneux, M., Gontrum, J., Fano, E., and Nivre, J. (2019). Deep Contextualized Word Embeddings in Transition-Based and Graph-Based Dependency Parsing - A Tale of Two Parsers Revisited. 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)*, pages 2755–2768, Hong Kong, China. Association for Computational Linguistics. Cited on pages 122, 123, 127, 128, and 129.

Kuncheva, L. I. and Whitaker, C. J. (2003). Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy. *Machine learning*, 51(2):181–207. Cited on page 65.

Kuncoro, A., Ballesteros, M., Kong, L., Dyer, C., and Smith, N. A. (2016). Distilling an Ensemble of Greedy Dependency Parsers into One MST Parser. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1744–1753, Austin, Texas. Association for Computational Linguistics. Cited on pages 4, 60, 68, 126, and 162.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998a). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324. Cited on page 55.

LeCun, Y., Bottou, L., Orr, G. B., and Müller, K. R. (1998b). *Efficient BackProp*, pages 9–50. Springer, Berlin, Heidelberg. Cited on page 55.

Lim, K., Park, C., Lee, C., and Poibeau, T. (2018). SEx BiST: A Multi-Source Trainable Parser with Deep Contextualized Lexical Representations. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 143–152, Brussels, Belgium. Association for Computational Linguistics. Cited on page 123.

Ling, W., Dyer, C., Black, A. W., and Trancoso, I. (2015). 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*, pages 1299–1304, Denver, Colorado. Association for Computational Linguistics. Cited on page 47.

Linzen, T., Dupoux, E., and Goldberg, Y. (2016). Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535. Cited on pages 148 and 161.

Lynn, T., Foster, J., Dras, M., and Tounsi, L. (2014). Cross-lingual Transfer Parsing for Low-Resourced Languages: An Irish Case Study. In *Proceedings of the First Celtic Language Technology Workshop*, pages 41–49, Dublin, Ireland. Association for Computational Linguistics and Dublin City University. Cited on pages 103 and 117.

Maamouri, M., Bies, A., Buckwalter, T., and Mekki, W. (2004). The Penn Arabic Treebank: Building a Large-Scale Annotated Arabic Corpus. In *NEMLAR Conference on Arabic Language Resources and Tools*. Cited on page 165.

Manning, C. D. (2015). Computational Linguistics and Deep Learning. *Computational Linguistics*, 41(4):701–707. Cited on page 6.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330. Cited on pages 16, 82, and 143.

Martins, A., Almeida, M., and Smith, N. A. (2013). Turning on the Turbo: Fast Third-Order Non-Projective Turbo Parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 617–622, Sofia, Bulgaria. Association for Computational Linguistics. Cited on pages 40 and 87.

Martins, A. F. T., Das, D., Smith, N. A., and Xing, E. P. (2008). Stacking Dependency Parsers. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 157–166, Honolulu, Hawaii. Association for Computational Linguistics. Cited on pages 61, 62, 65, 85, 88, and 93.

McDonald, R., Crammer, K., and Pereira, F. (2005a). Online Large-Margin Training of Dependency Parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 91–98, Ann Arbor, Michigan, USA. Association for Computational Linguistics. Cited on pages xiii, 30, 33, 38, and 125.

McDonald, R., Lerman, K., and Pereira, F. (2006). Multilingual Dependency Analysis with a Two-Stage Discriminative Parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 216–220, New York City. Association for Computational Linguistics. Cited on page 69.

McDonald, R. and Nivre, J. (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)*, pages 122–131, Prague, Czech Republic. Association for Computational Linguistics. Cited on pages 57, 89, 136, and 145.

McDonald, R., Nivre, J., Quirmbach-Brundage, Y., Goldberg, Y., Das, D., Ganchev, K., Hall, K., Petrov, S., Zhang, H., Täckström, O., Bedini, C., Bertomeu Castelló, N., and Lee, J. (2013). Universal Dependency Annotation for Multilingual Parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 92–97, Sofia, Bulgaria. Association for Computational Linguistics. Cited on page 74.

McDonald, R. and Pereira, F. (2006). Online Learning of Approximate Dependency Parsing Algorithms. In *Proceedings of the 11th Conference of the European Chapter of the ACL (EACL 2006)*, pages 81–88, Trento, Italy. Association for Computational Linguistics. Cited on pages 40, 111, and 143.

McDonald, R., Pereira, F., Ribarov, K., and Hajic, J. (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*, pages 523–530, Vancouver, British Columbia, Canada. Association for Computational Linguistics. Cited on page 34.

McDonald, R., Petrov, S., and Hall, K. (2011). Multi-Source Transfer of Delexicalized Dependency Parsers. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 62–72, Edinburgh, Scotland, UK. Association for Computational Linguistics. Cited on pages 103 and 106.

McDonald, R. and Satta, G. (2007). On the Complexity of Non-Projective Data-Driven Dependency Parsing. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 121–132, Prague, Czech Republic. Association for Computational Linguistics. Cited on page 40.

Mel'čuk, I. (1988). *Dependency Syntax: Theory and Practice*. State University Press of New York, SUNY serie edition. Cited on page 13.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*. Cited on page 47.

Mirroshandel, S. A. and Nasr, A. (2011). Active Learning for Dependency Parsing Using Partially Annotated Sentences. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 140–149, Dublin, Ireland. Association for Computational Linguistics. Cited on pages 74 and 75.

Mueller, T., Schmid, H., and Schütze, H. (2013). Efficient Higher-Order CRFs for Morphological Tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 322–332, Seattle, Washington, USA. Association for Computational Linguistics. Cited on pages 83, 110, 127, and 144.

Neubig, G., Dyer, C., Goldberg, Y., Matthews, A., Ammar, W., Anastasopoulos, A., Ballesteros, M., Chiang, D., Clothiaux, D., Cohn, T., Duh, K., Faruqui, M., Gan, C., Garrette, D., Ji, Y., Kong, L., Kuncoro, A., Kumar, G., Malaviya, C., Michel, P., Oda, Y., Richardson, M., Saphra, N., Swayamdipta, S., and Yin, P. (2017). DyNet: The Dynamic Neural Network Toolkit. *arXiv preprint arXiv:1701.03980*. Cited on pages 123 and 144.

Ninomiya, T., Matsuzaki, T., Tsuruoka, Y., Miyao, Y., and Tsujii, J. (2006). Extremely Lexicalized Models for Accurate and Fast HPSG Parsing. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 155–163, Sydney, Australia. Association for Computational Linguistics. Cited on page 80.

Nivre, J. (2003). An Efficient Algorithm for Projective Dependency Parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*, pages 149–160. Cited on pages 22, 23, and 29.

Nivre, J. (2004). Incrementality in Deterministic Dependency Parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57, Barcelona, Spain. Association for Computational Linguistics. Cited on page 23.

Nivre, J. (2008). Algorithms for Deterministic Incremental Dependency Parsing. *Computational Linguistics*, 34(4):513–553. Cited on page 23.

Nivre, J. (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*, pages 351–359, Suntec, Singapore. Association for Computational Linguistics. Cited on pages xiii, 23, 24, 25, 27, 85, 111, 124, and 141.

Nivre, J., Abrams, M., Agić, Ž., Ahrenberg, L., Aleksandravičiūtė, G., Antonsen, L., Aplonova, K., Aranzabe, M. J., Arutie, G., Asahara, M., Ateyah, L., Attia, M., Atutxa,

A., Augustinus, L., Badmaeva, E., Ballesteros, M., Banerjee, E., Bank, S., Barbu Mititelu, V., Basmov, V., Bauer, J., Bellato, S., Bengoetxea, K., Berzak, Y., Bhat, I. A., Bhat, R. A., Biagetti, E., Bick, E., Bielinskienė, A., Blokland, R., Bobicev, V., Boizou, L., Borges Völker, E., Börstell, C., Bosco, C., Bouma, G., Bowman, S., Boyd, A., Brokaitė, K., Burchardt, A., Candito, M., Caron, B., Caron, G., Cebiroğlu Eryiğit, G., Cecchini, F. M., Celano, G. G. A., Čéplö, S., Cetin, S., Chalub, F., Choi, J., Cho, Y., Chun, J., Cinková, S., Collomb, A., Çöltekin, Ç., Connor, M., Courtin, M., Davidson, E., de Marneffe, M.-C., de Paiva, V., Diaz de Ilarraza, A., Dickerson, C., Dione, B., Dirix, P., Dobrovoljc, K., Dozat, T., Droganova, K., Dwivedi, P., Eckhoff, H., Eli, M., Elkahky, A., Ephrem, B., Erjavec, T., Etienne, A., Farkas, R., Fernandez Alcalde, H., Foster, J., Freitas, C., Fujita, K., Gajdošová, K., Galbraith, D., Garcia, M., Gärdenfors, M., Garza, S., Gerdes, K., Ginter, F., Goenaga, I., Gojenola, K., Gökırmak, M., Goldberg, Y., Gómez Guinovart, X., González Saavedra, B., Grioni, M., Grūzītis, N., Guillaume, B., Guillot-Barbance, C., Habash, N., Hajič, J., Hajič jr., J., Hà Mỹ, L., Han, N.-R., Harris, K., Haug, D., Heinecke, J., Hennig, F., Hladká, B., Hlaváčová, J., Hociung, F., Hohle, P., Hwang, J., Ikeda, T., Ion, R., Irimia, E., Ishola, Ọ., Jelínek, T., Johannsen, A., Jørgensen, F., Kaşıkara, H., Kaasen, A., Kahane, S., Kanayama, H., Kanerva, J., Katz, B., Kayadelen, T., Kenney, J., Kettnerová, V., Kirchner, J., Köhn, A., Kopacewicz, K., Kotsyba, N., Kovalevskaitė, J., Krek, S., Kwak, S., Laippala, V., Lambertino, L., Lam, L., Lando, T., Larasati, S. D., Lavrentiev, A., Lee, J., Lê Hồng, P., Lenci, A., Lertpradit, S., Leung, H., Li, C. Y., Li, J., Li, K., Lim, K., Li, Y., Ljubešić, N., Loginova, O., Lyashevskaya, O., Lynn, T., Macketanz, V., Makazhanov, A., Mandl, M., Manning, C., Manurung, R., Mărănduc, C., Mareček, D., Marheinecke, K., Martínez Alonso, H., Martins, A., Mašek, J., Matsumoto, Y., McDonald, R., McGuinness, S., Mendonça, G., Miekka, N., Misirpashayeva, M., Missilä, A., Mititelu, C., Miyao, Y., Montemagni, S., More, A., Moreno Romero, L., Mori, K. S., Morioka, T., Mori, S., Moro, S., Mortensen, B., Moskalevskyi, B., Muischnek, K., Murawaki, Y., Müürisep, K., Nainwani, P., Navarro Horñiacek, J. I., Nedoluzhko, A., Nešpore-Bērzkalne, G., Nguyễn Thị, L., Nguyễn Thị Minh, H., Nikaido, Y., Nikolaev, V., Nitisaroj, R., Nurmi, H., Ojala, S., Olúòkun, A., Omura, M., Osenova, P., Östling, R., Øvrelid, L., Partanen, N., Pascual, E., Passarotti, M., Patejuk, A., Paulino-Passos, G., Peljak-Łapińska, A., Peng, S., Perez, C.-A., Perrier, G., Petrova, D., Petrov, S., Piitulainen, J., Pirinen, T. A., Pitler, E., Plank, B., Poibeau, T., Popel, M., Pretkalniņa, L., Prévost, S., Prokopidis, P., Przepiórkowski, A., Puolakainen, T., Pyysalo, S., Rääbis, A., Rademaker, A., Ramasamy, L., Rama, T., Ramisch, C., Ravishankar, V., Real, L., Reddy, S., Rehm, G., Rießler, M., Rimkutė, E., Rinaldi, L., Rituma, L., Rocha, L., Romanenko, M., Rosa, R., Rovati, D., Roșca, V., Rudina, O., Rueter, J., Sadde, S., Sagot,

B., Saleh, S., Salomoni, A., Samardžić, T., Samson, S., Sanguinetti, M., Särg, D., Saulīte, B., Sawanakunanon, Y., Schneider, N., Schuster, S., Seddah, D., Seeker, W., Seraji, M., Shen, M., Shimada, A., Shirasu, H., Shohibussirri, M., Sichinava, D., Silveira, N., Simi, M., Simionescu, R., Simkó, K., Šimková, M., Simov, K., Smith, A., Soares-Bastos, I., Spadine, C., Stella, A., Straka, M., Strnadová, J., Suhr, A., Sulubacak, U., Suzuki, S., Szántó, Z., Taji, D., Takahashi, Y., Tamburini, F., Tanaka, T., Tellier, I., Thomas, G., Torga, L., Trosterud, T., Trukhina, A., Tsarfaty, R., Tyers, F., Uematsu, S., Urešová, Z., Uria, L., Uszkoreit, H., Vajjala, S., van Niekerk, D., van Noord, G., Varga, V., Villemonte de la Clergerie, E., Vincze, V., Wallin, L., Walsh, A., Wang, J. X., Washington, J. N., Wendt, M., Williams, S., Wirén, M., Wittern, C., Woldemariam, T., Wong, T.-s., Wróblewska, A., Yako, M., Yamazaki, N., Yan, C., Yasuoka, K., Yavrumyan, M. M., Yu, Z., Žabokrtský, Z., Zeldes, A., Zeman, D., Zhang, M., and Zhu, H. (2019). Universal Dependencies 2.4. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University. Cited on pages 18, 127, and 168.

Nivre, J., Agić, Ž., Ahrenberg, L., Aranzabe, M. J., Asahara, M., Atutxa, A., Ballesteros, M., Bauer, J., Bengoetxea, K., Bhat, R. A., Bick, E., Bosco, C., Bouma, G., Bowman, S., Candito, M., Cebiroğlu Eryiğit, G., Celano, G. G. A., Chalub, F., Choi, J., Çöltekin, Ç., Connor, M., Davidson, E., de Marneffe, M.-C., de Paiva, V., Diaz de Ilarraza, A., Dobrovoljc, K., Dozat, T., Droganova, K., Dwivedi, P., Eli, M., Erjavec, T., Farkas, R., Foster, J., Freitas, C., Gajdošová, K., Galbraith, D., Garcia, M., Ginter, F., Goenaga, I., Gojenola, K., Gökırmak, M., Goldberg, Y., Gómez Guinovart, X., Gonzáles Saavedra, B., Grioni, M., Grūzītis, N., Guillaume, B., Habash, N., Hajič, J., Hà Mỹ, L., Haug, D., Hladká, B., Hohle, P., Ion, R., Irimia, E., Johannsen, A., Jørgensen, F., Kaşıkara, H., Kanayama, H., Kanerva, J., Kotsyba, N., Krek, S., Laippala, V., Lê Hồng, P., Lenci, A., Ljubešić, N., Lyashevskaya, O., Lynn, T., Makazhanov, A., Manning, C., Mărănduc, C., Mareček, D., Martínez Alonso, H., Martins, A., Mašek, J., Matsumoto, Y., McDonald, R., Missilä, A., Mititelu, V., Miyao, Y., Montemagni, S., More, A., Mori, S., Moskalevskyi, B., Muischnek, K., Mustafina, N., Müürisep, K., Nguyễn Thị, L., Nguyễn Thị Minh, H., Nikolaev, V., Nurmi, H., Ojala, S., Osenova, P., Øvrelid, L., Pascual, E., Passarotti, M., Perez, C.-A., Perrier, G., Petrov, S., Piitulainen, J., Plank, B., Popel, M., Pretkalniņa, L., Prokopidis, P., Puolakainen, T., Pyysalo, S., Rademaker, A., Ramasamy, L., Real, L., Rituma, L., Rosa, R., Saleh, S., Sanguinetti, M., Saulīte, B., Schuster, S., Seddah, D., Seeker, W., Seraji, M., Shakurova, L., Shen, M., Sichinava, D., Silveira, N., Simi, M., Simionescu, R., Simkó, K., Šimková, M., Simov, K., Smith, A., Suhr, A., Sulubacak, U.,

Szántó, Z., Taji, D., Tanaka, T., Tsarfaty, R., Tyers, F., Uematsu, S., Uria, L., van Noord, G., Varga, V., Vincze, V., Washington, J. N., Žabokrtský, Z., Zeldes, A., Zeman, D., and Zhu, H. (2017). Universal Dependencies 2.0. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University. Cited on pages 18, 103, 108, 143, and 168.

Nivre, J., de Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajič, J., Manning, C. D., McDonald, R., Petrov, S., Pyysalo, S., Silveira, N., Tsarfaty, R., and Zeman, D. (2016). Universal Dependencies v1: A Multilingual Treebank Collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1659–1666, Portorož, Slovenia. European Language Resources Association (ELRA). Cited on pages 14, 18, and 102.

Nivre, J., Hall, J., Kübler, S., McDonald, R., Nilsson, J., Riedel, S., and Yuret, D. (2007). The CoNLL 2007 Shared Task on Dependency Parsing. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 915–932, Prague, Czech Republic. Association for Computational Linguistics. Cited on pages 65 and 68.

Nivre, J., Hall, J., Nilsson, J., Eryiğit, G., and Marinov, S. (2006a). Labeled Pseudo-Projective Dependency Parsing with Support Vector Machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 221–225, New York City. Association for Computational Linguistics. Cited on page 69.

Nivre, J., Kuhlmann, M., and Hall, J. (2009). An Improved Oracle for Dependency Parsing with Online Reordering. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 73–76, Paris, France. Association for Computational Linguistics. Cited on pages xv, 25, 27, 85, 111, 124, and 141.

Nivre, J. and McDonald, R. (2008). Integrating Graph-Based and Transition-Based Dependency Parsers. In *Proceedings of ACL-08: HLT*, pages 950–958, Columbus, Ohio. Association for Computational Linguistics. Cited on pages xiii, 3, 61, 62, 79, 85, 87, 88, and 160.

Nivre, J., Nilsson, J., and Hall, J. (2006b). Talbanken05: A Swedish Treebank with Phrase Structure and Dependency Annotation. In *Proceedings of LREC*, pages 1392–1395, Genoa, Italy. Cited on page 165.

Oepen, S., Øvrelid, L., Björne, J., Johansson, R., Lapponi, E., Ginter, F., and Velldal, E. (2017). The 2017 Shared Task on Extrinsic Parser Evaluation Towards a Reusable Community Infrastructure. In *Proceedings of the 2017 Shared Task on Extrinsic Parser Evaluation*, pages 1–16. Cited on pages 21 and 71.

O'Horan, H., Berzak, Y., Vulić, I., Reichart, R., and Korhonen, A. (2016). Survey on the Use of Typological Information in Natural Language Processing. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1297–1308, Osaka, Japan. The COLING 2016 Organizing Committee. Cited on page 103.

Ouchi, H., Duh, K., and Matsumoto, Y. (2014). Improving Dependency Parsers with Supertags. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, volume 2: Short Papers*, pages 154–158, Gothenburg, Sweden. Association for Computational Linguistics. Cited on pages 80, 81, 83, 85, 88, 94, 99, and 126.

Øvrelid, L., Kuhn, J., and Spreyer, K. (2009a). Cross-framework parser stacking for data-driven dependency parsing. *Trait. Autom. des Langues*, 50(3):109–138. Cited on page 66.

Øvrelid, L., Kuhn, J., and Spreyer, K. (2009b). Improving data-driven dependency parsing using large-scale LFG grammars. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 37–40, Suntec, Singapore. Association for Computational Linguistics. Cited on pages 66 and 100.

Palmer, M., Gildea, D., and Kingsbury, P. (2005). The Proposition Bank: An Annotated Corpus of Semantic Roles. *Computational Linguistics*, 31(1):71–106. Cited on page 22.

Park, S., Choi, D., Kim, E., and Choi, K. (2010). A Plug-In Component-based Korean Morphological Analyzer. In *Annual Conference on Human and Language Technology*, pages 197–201. Human and Language Technology. Cited on page 84.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training Recurrent Neural Networks. In *International conference on machine learning*, pages 1310–1318. Cited on page 48.

Pei, W., Ge, T., and Chang, B. (2015). An Effective Neural Network Model for Graph-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), pages 313–322, Beijing, China. Association for Computational Linguistics. Cited on pages ix, 45, 46, 47, and 50.

Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (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)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics. Cited on pages 52, 121, and 123.

Petrov, S., Das, D., and McDonald, R. (2012). A Universal Part-of-Speech Tagset. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 2089–2096, Istanbul, Turkey. European Language Resources Association (ELRA). Cited on page 18.

Petrov, S. and McDonald, R. (2012). Overview of the 2012 Shared Task on Parsing the Web. In *Notes of Syntactic Analysis of Non-Canonical Language (SANCL)*, volume 59. Cited on page 68.

Plank, B., Søgaard, A., and Goldberg, Y. (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)*, pages 412–418, Berlin, Germany. Association for Computational Linguistics. Cited on pages 6 and 83.

Polikar, R. (2006). Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, 6(3):21–45. Cited on page 3.

Popel, M. and Žabokrtskỳ, Z. (2010). TectoMT: modular NLP framework. In *International Conference on Natural Language Processing*, pages 293–304. Springer. Cited on page 72.

Rebholz-Schuhmann, D., Jimeno Yepes, A. J., van Mulligen, E. M., Kang, N., Kors, J., Milward, D., Corbett, P., Buyko, E., Tomanek, K., Beisswanger, E., and Hahn, U. (2010). The CALBC Silver Standard Corpus for Biomedical Named Entities — A Study in Harmonizing the Contributions from Four Independent Named Entity Taggers. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta. European Language Resources Association (ELRA). Cited on page 75.

Reimers, N. and Gurevych, I. (2017). Reporting Score Distributions Makes a Difference: Performance Study of LSTM-networks for Sequence Tagging. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 338–348,

Copenhagen, Denmark. Association for Computational Linguistics. Cited on pages 67 and 131.

Reimers, N. and Gurevych, I. (2018). Why Comparing Single Performance Scores Does Not Allow to Draw Conclusions About Machine Learning Approaches. *arXiv preprint arXiv:1803.09578*. Cited on pages 127 and 144.

Rosa, R. and Žabokrtský, Z. (2015). KLcpos3 - a Language Similarity Measure for Delexicalized Parser Transfer. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 243–249, Beijing, China. Association for Computational Linguistics. Cited on pages 4, 103, 104, 106, 107, and 108.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386. Cited on page 54.

Rowling, J. K. (2000). *Harry Potter and the Chamber of Secrets*. New York : Scholastic, Inc. Cited on page 1.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536. Cited on page 55.

Rush, A. and Petrov, S. (2012). Vine Pruning for Efficient Multi-Pass Dependency Parsing. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 498–507, Montréal, Canada. Association for Computational Linguistics. Cited on page 40.

Sagae, K. and Lavie, A. (2006). Parser Combination by Reparsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 129–132, New York City, USA. Association for Computational Linguistics. Cited on pages 3, 63, 64, and 65.

Sagae, K. and Tsujii, J. (2007). Dependency Parsing and Domain Adaptation with LR Models and Parser Ensembles. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 1044–1050, Prague, Czech Republic. Association for Computational Linguistics. Cited on page 67.

Sangati, F., Zuidema, W., and Bod, R. (2009). A generative re-ranking model for dependency parsing. In *Proceedings of the 11th International Conference on Parsing Technologies*

(IWPT'09), pages 238–241, Paris, France. Association for Computational Linguistics. Cited on page 67.

Sartorio, F., Satta, G., and Nivre, J. (2013). A Transition-Based Dependency Parser Using a Dynamic Parsing Strategy. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 135–144, Sofia, Bulgaria. Association for Computational Linguistics. Cited on page 23.

Schabes, Y., Abeille, A., and Joshi, A. K. (1988). Parsing Strategies with 'Lexicalized' Grammars: Application to Tree Adjoining Grammars. In *Proceedings of the 12th Conference on Computational Linguistics - Volume 2*, COLING '88, pages 578–583, Stroudsburg, PA, USA. Association for Computational Linguistics. Cited on page 80.

Schmid, H., Fitschen, A., and Heid, U. (2004). SMOR: A German Computational Morphology Covering Derivation, Composition, and Inflection. In *Proceedings of the IVth International Conference on Language Resources and Evaluation (LREC 2004*, pages 1263–1266. Cited on page 84.

Schuster, S., Villemonte de La Clergerie, É., Candito, M. D., Sagot, B., Manning, C. D., and Seddah, D. (2017). Paris and Stanford at EPE 2017: Downstream Evaluation of Graph-based Dependency Representations. In *EPE 2017 - The First Shared Task on Extrinsic Parser Evaluation*, Proceedings of the 2017 Shared Task on Extrinsic Parser Evaluation, pages 47–59, Pisa, Italy. Cited on page 72.

Schuster, T., Ram, O., Barzilay, R., and Globerson, A. (2019). Cross-Lingual Alignment of Contextual Word Embeddings, with Applications to Zero-shot Dependency Parsing. 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)*, pages 1599–1613, Minneapolis, Minnesota. Association for Computational Linguistics. Cited on page 123.

Schweitzer, K., Eckart, K., Gärtner, M., Falenska, A., Riester, A., Rösiger, I., Schweitzer, A., Stehwien, S., and Kuhn, J. (2018). German Radio Interviews: The GRAIN Release of the SFB732 Silver Standard Collection. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation*, pages 2887–2895, Miyazaki, Japan. European Language Resources Association. Cited on pages 4, 63, and 75.

Seddah, D., Tsarfaty, R., Kübler, S., Candito, M., Choi, J., Constant, M., Farkas, R., Goenaga, I., Gojenola, K., Goldberg, Y., Green, S., Habash, N., Kuhlmann, M., Maier, W.,

Nivre, J., Przepiorkowski, A., Roth, R., Seeker, W., Versley, Y., Vincze, V., Woliński, M., Wróblewska, A., and Villemonte de la Clérgerie, E. (2014). Overview of the SPMRL 2014 Shared Task on Parsing Morphologically Rich Languages. In *Notes of the SPMRL 2014 Shared Task on Parsing Morphologically-Rich Languages*, Dublin, Ireland. Cited on pages 68, 82, 165, and 166.

Seddah, D., Tsarfaty, R., Kübler, S., Candito, M., Choi, J. D., Farkas, R., Foster, J., Goenaga, I., Gojenola Galletebeitia, K., Goldberg, Y., Green, S., Habash, N., Kuhlmann, M., Maier, W., Nivre, J., Przepiórkowski, A., Roth, R., Seeker, W., Versley, Y., Vincze, V., Woliński, M., Wróblewska, A., and Villemonte de la Clergerie, E. (2013). Overview of the SPMRL 2013 Shared Task: A Cross-Framework Evaluation of Parsing Morphologically Rich Languages. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 146–182, Seattle, Washington, USA. Association for Computational Linguistics. Cited on page 68.

Seeker, W. and Kuhn, J. (2012). Making Ellipses Explicit in Dependency Conversion for a German Treebank. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 3132–3139, Istanbul, Turkey. European Language Resources Association (ELRA). Cited on pages 17 and 165.

Sekine, S. (1997). The Domain Dependence of Parsing. In *Fifth Conference on Applied Natural Language Processing*, pages 96–102, Washington, DC, USA. Association for Computational Linguistics. Cited on page 74.

Sgall, P., Hajičová, E., and Panevová, J. (1986). *The Meaning of the Sentence in Its Semantic and Pragmatic Aspects*. Dordrecht:Reidel Publishing Company and Prague:Academia. Cited on page 13.

Shi, T., Huang, L., and Lee, L. (2017a). Fast(er) Exact Decoding and Global Training for Transition-Based Dependency Parsing via a Minimal Feature Set. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 12–23. Association for Computational Linguistics. Cited on pages 52, 125, 142, 146, and 155.

Shi, T., Wu, F. G., Chen, X., and Cheng, Y. (2017b). Combining Global Models for Parsing Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 31–39, Vancouver, Canada. Association for Computational Linguistics. Cited on pages 4 and 127.

Shi, X., Padhi, I., and Knight, K. (2016). Does String-Based Neural MT Learn Source Syntax? In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1526–1534. Association for Computational Linguistics. Cited on page 161.

Sima'an, K., Itai, A., Winter, Y., Altman, A., and Nativ, N. (2001). Building a Tree-Bank for Modern Hebrew Text. In *Traitement Automatique des Langues*, volume 42(2). Cited on page 165.

Smith, A., de Lhoneux, M., Stymne, S., and Nivre, J. (2018). An Investigation of the Interactions Between Pre-Trained Word Embeddings, Character Models and POS Tags in Dependency Parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2711–2720. Association for Computational Linguistics. Cited on page 143.

Søgaard, A. and Goldberg, Y. (2016). Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 231–235, Berlin, Germany. Association for Computational Linguistics. Cited on page 127.

Søgaard, A. and Rishøj, C. (2010). Semi-supervised dependency parsing using generalized tri-training. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 1065–1073, Beijing, China. Coling 2010 Organizing Committee. Cited on page 67.

Sohl, J. and Zinsmeister, H. (2018). Exploring Ensemble Dependency Parsing to Reduce Manual Annotation Workload. In Rehm, G. and Declerck, T., editors, *Language Technologies for the Challenges of the Digital Age*, pages 40–47, Cham. Springer International Publishing. Cited on page 74.

Souček, M., Järvinen, T., and LaMontagne, A. (2013). Managing a Multilingual Treebank Project. In *Proceedings of the Second International Conference on Dependency Linguistics (DepLing 2013)*, pages 292–297, Prague, Czech Republic. Charles University in Prague, Matfyzpress, Prague, Czech Republic. Cited on pages 5, 73, and 105.

Spreyer, K. and Kuhn, J. (2009). Data-Driven Dependency Parsing of New Languages Using Incomplete and Noisy Training Data. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009)*, pages 12–20, Boulder, Colorado. Association for Computational Linguistics. Cited on page 67.

Strzyz, M., Vilares, D., and Gómez-Rodríguez, C. (2019). Viable Dependency Parsing as Sequence Labeling. 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)*, pages 717–723, Minneapolis, Minnesota. Association for Computational Linguistics. Cited on pages 138 and 162.

Surdeanu, M., Johansson, R., Meyers, A., Màrquez, L., and Nivre, J. (2008). The CoNLL 2008 Shared Task on Joint Parsing of Syntactic and Semantic Dependencies. In *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 159–177, Manchester, England. Coling 2008 Organizing Committee. Cited on page 68.

Surdeanu, M. and Manning, C. D. (2010). Ensemble Models for Dependency Parsing: Cheap and Good? In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 649–652, Los Angeles, California. Association for Computational Linguistics. Cited on pages 63, 64, 65, 74, 75, 82, 93, and 99.

Świdziński, M. and Woliński, M. (2010). Towards a Bank of Constituent Parse Trees for Polish. In *Text, Speech and Dialogue: 13th International Conference (TSD)*, Lecture Notes in Computer Science, pages 197—204, Brno, Czech Republic. Springer. Cited on pages 17, 74, and 165.

Sylak-Glassman, J., Kirov, C., Yarowsky, D., and Que, R. (2015). A Language-Independent Feature Schema for Inflectional Morphology. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 674–680, Beijing, China. Association for Computational Linguistics. Cited on page 18.

Täckström, O., McDonald, R., and Uszkoreit, J. (2012). Cross-lingual Word Clusters for Direct Transfer of Linguistic Structure. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL HLT '12, pages 477–487, Montreal, Canada. Cited on page 104.

Tarjan, R. E. (1977). Finding optimum branchings. *Networks*, 7(1):25–35. Cited on page 34.

Tesnière, L. (1959). Eléments de syntaxe structurale. *Paris, Klincksieck*. Cited on page 13.

Tesnière, L. (2015). *Elements of Structural Syntax*. John Benjamins. Cited on page 13.

Tiedemann, J. (2012). Parallel Data, Tools and Interfaces in OPUS. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 2214–2218, Istanbul, Turkey. European Language Resources Association (ELRA). Cited on page 103.

Tiedemann, J. and Agić, Ž. (2016). Synthetic Treebanking for Cross-Lingual Dependency Parsing. *Journal of AI Research*, 55(1):209–248. Cited on page 104.

Titov, I. and Henderson, J. (2007). A latent variable model for generative dependency parsing. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 144–155, Prague, Czech Republic. Association for Computational Linguistics. Cited on page 41.

Tsarfaty, R. (2010). *Relational-Realizational Parsing*. PhD thesis, University of Amsterdam. Cited on page 165.

Tsarfaty, R. (2013). *A Unified Morpho-Syntactic Scheme of Stanford Dependencies*. Proceedings of ACL. Cited on page 165.

Vincze, V., Szauter, D., Almási, A., Móra, G., Alexin, Z., and Csirik, J. (2010). Hungarian Dependency Treebank. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta. European Language Resources Association (ELRA). Cited on page 165.

Wang, W. and Chang, B. (2016). Graph-based Dependency Parsing with Bidirectional LSTM. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2306–2315, Berlin, Germany. Association for Computational Linguistics. Cited on pages 52 and 147.

Wang, Y., Johnson, M., Wan, S., Sun, Y., and Wang, W. (2019). How to Best Use Syntax in Semantic Role Labelling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5338–5343, Florence, Italy. Association for Computational Linguistics. Cited on page 6.

Woliński, M. (2006). Morfeusz - a Practical Tool for the Morphological Analysis of Polish. In *Intelligent Information Processing and Web Mining*, pages 511–520. Springer. Cited on page 84.

Woliński, M., Głowińska, K., and Świdziński, M. (2011). A preliminary version of Składnica—a treebank of Polish. In *Proceedings of the 5th Language & Technology Conference, Poznań*, pages 299–303. Cited on page 17.

Wolpert, D. H. (1992). Stacked generalization. *Neural networks*, 5(2):241–259. Cited on page 60.

Wróblewska, A. (2018). Extended and Enhanced Polish Dependency Bank in Universal Dependencies Format. In *Proceedings of the Second Workshop on Universal Dependencies (UDW 2018)*, pages 173–182, Brussels, Belgium. Association for Computational Linguistics. Cited on page 72.

Yamada, H. and Matsumoto, Y. (2003). Statistical Dependency Analysis with Support Vector Machines. In *Proceedings of the Eighth International Conference on Parsing Technologies*, pages 195–206, Nancy, France. Cited on pages 22 and 41.

Yarowsky, D. (1995). Unsupervised Word Sense Disambiguation Rivaling Supervised Methods. In *33rd Annual Meeting of the Association for Computational Linguistics*, pages 189–196, Cambridge, Massachusetts, USA. Association for Computational Linguistics. Cited on page 66.

Younger, D. H. (1967). Recognition and parsing of context-free languages in time n3. *Information and control*, 10(2):189–208. Cited on page 38.

Yu, X., Falenska, A., Haid, M., Vu, N. T., and Kuhn, J. (2019a). IMSurReal: IMS at the Surface Realization Shared Task 2019. In *Proceedings of the 2nd Workshop on Multilingual Surface Realisation*, pages 50–58, Hong Kong, China. Association for Computational Linguistics.

Yu, X., Falenska, A., and Kuhn, J. (2019b). Dependency Length Minimization vs. Word Order Constraints: An Empirical Study On 55 Treebanks. In *Proceedings of the First Workshop on Quantitative Syntax*, pages 89–97, Paris, France. Association for Computational Linguistics.

Yu, X., Falenska, A., and Vu, N. T. (2017). A General-Purpose Tagger with Convolutional Neural Networks. In *Proceedings of the First Workshop on Subword and Character Level Models in NLP*, pages 124–129, Copenhagen, Denmark. Association for Computational Linguistics.

Yu, X., Falenska, A., Vu, N. T., and Kuhn, J. (2019c). Head-First Linearization with Tree-Structured Representation. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 279–289, Tokyo, Japan. Association for Computational Linguistics.

Zeman, D., Hajič, J., Popel, M., Potthast, M., Straka, M., Ginter, F., Nivre, J., and Petrov, S. (2018). CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21, Brussels, Belgium. Association for Computational Linguistics. Cited on pages 4, 51, 67, 69, 122, and 140.

Zeman, D., Nivre, J., Abrams, M., Ackermann, E., Aepli, N., Agić, Ž., Ahrenberg, L., Ajede, C. K., Aleksandravičiūtė, G., Antonsen, L., Aplonova, K., Aquino, A., Aranzabe, M. J., Arutie, G., Asahara, M., Ateyah, L., Atmaca, F., Attia, M., Atutxa, A., Augustinus, L., Badmaeva, E., Ballesteros, M., Banerjee, E., Bank, S., Barbu Mititelu, V., Basmov, V., Batchelor, C., Bauer, J., Bengoetxea, K., Berzak, Y., Bhat, I. A., Bhat, R. A., Biagetti, E., Bick, E., Bielinskienė, A., Blokland, R., Bobicev, V., Boizou, L., Borges Völker, E., Börstell, C., Bosco, C., Bouma, G., Bowman, S., Boyd, A., Brokaitė, K., Burchardt, A., Candito, M., Caron, B., Caron, G., Cavalcanti, T., Cebiroğlu Eryiğit, G., Cecchini, F. M., Celano, G. G. A., Čéplö, S., Cetin, S., Chalub, F., Chi, E., Choi, J., Cho, Y., Chun, J., Cignarella, A. T., Cinková, S., Collomb, A., Çöltekin, Ç., Connor, M., Courtin, M., Davidson, E., de Marneffe, M.-C., de Paiva, V., de Souza, E., Diaz de Ilarraza, A., Dickerson, C., Dione, B., Dirix, P., Dobrovoljc, K., Dozat, T., Droganova, K., Dwivedi, P., Eckhoff, H., Eli, M., Elkahky, A., Ephrem, B., Erina, O., Erjavec, T., Etienne, A., Evelyn, W., Farkas, R., Fernandez Alcalde, H., Foster, J., Freitas, C., Fujita, K., Gajdošová, K., Galbraith, D., Garcia, M., Gärdenfors, M., Garza, S., Gerdes, K., Ginter, F., Goenaga, I., Gojenola, K., Gökırmak, M., Goldberg, Y., Gómez Guinovart, X., González Saavedra, B., Griciūtė, B., Grioni, M., Grobol, L., Grūzītis, N., Guillaume, B., Guillot-Barbance, C., Güngör, T., Habash, N., Hajič, J., Hajič jr., J., Hämäläinen, M., Hà Mỹ, L., Han, N.-R., Harris, K., Haug, D., Heinecke, J., Hellwig, O., Hennig, F., Hladká, B., Hlaváčová, J., Hociung, F., Hohle, P., Hwang, J., Ikeda, T., Ion, R., Irimia, E., Ishola, Ọ., Jelínek, T., Johannsen, A., Jónsdóttir, H., Jørgensen, F., Juutinen, M., Kaşıkara, H., Kaasen, A., Kabaeva, N., Kahane, S., Kanayama, H., Kanerva, J., Katz, B., Kayadelen, T., Kenney, J., Kettnerová, V., Kirchner, J., Klementieva, E., Köhn, A., Köksal, A., Kopacewicz, K., Korkiakangas, T., Kotsyba, N., Kovalevskaitė, J., Krek, S., Kwak, S., Laippala, V., Lambertino, L., Lam, L., Lando, T., Larasati, S. D., Lavrentiev, A., Lee, J., Lê Hồng, P., Lenci, A., Lertpradit, S., Leung, H., Levina, M., Li, C. Y., Li, J., Li, K., Lim, K., Li, Y., Ljubešić, N., Loginova, O., Lyashevskaya, O., Lynn, T., Macketanz, V., Makazhanov, A., Mandl, M., Manning, C., Manurung, R., Mărănduc, C., Mareček, D., Marheinecke, K., Martínez Alonso, H., Martins, A., Mašek, J., Matsuda, H., Matsumoto, Y., McDonald, R., McGuinness, S., Mendonça, G., Miekka, N., Misirpashayeva, M., Missilä, A.,

Mititelu, C., Mitrofan, M., Miyao, Y., Montemagni, S., More, A., Moreno Romero, L., Mori, K. S., Morioka, T., Mori, S., Moro, S., Mortensen, B., Moskalevskyi, B., Muischnek, K., Munro, R., Murawaki, Y., Müürisep, K., Nainwani, P., Navarro Horñiacek, J. I., Nedoluzhko, A., Nešpore-Bērzkalne, G., Nguyễn Thị, L., Nguyễn Thị Minh, H., Nikaido, Y., Nikolaev, V., Nitisaroj, R., Nurmi, H., Ojala, S., Ojha, A. K., Olúòkun, A., Omura, M., Onwuegbuzia, E., Osenova, P., Östling, R., Øvrelid, L., Özateş, Ş. B., Özgür, A., Öztürk Başaran, B., Partanen, N., Pascual, E., Passarotti, M., Patejuk, A., Paulino-Passos, G., Peljak-Łapińska, A., Peng, S., Perez, C.-A., Perrier, G., Petrova, D., Petrov, S., Phelan, J., Piitulainen, J., Pirinen, T. A., Pitler, E., Plank, B., Poibeau, T., Ponomareva, L., Popel, M., Pretkalniņa, L., Prévost, S., Prokopidis, P., Przepiórkowski, A., Puolakainen, T., Pyysalo, S., Qi, P., Rääbis, A., Rademaker, A., Ramasamy, L., Rama, T., Ramisch, C., Ravishankar, V., Real, L., Rebeja, P., Reddy, S., Rehm, G., Riabov, I., Rießler, M., Rimkutė, E., Rinaldi, L., Rituma, L., Rocha, L., Romanenko, M., Rosa, R., Roșca, V., Rovati, D., Rudina, O., Rueter, J., Sadde, S., Sagot, B., Saleh, S., Salomoni, A., Samardžić, T., Samson, S., Sanguinetti, M., Särg, D., Saulīte, B., Sawanakunanon, Y., Scarlata, S., Schneider, N., Schuster, S., Seddah, D., Seeker, W., Seraji, M., Shen, M., Shimada, A., Shirasu, H., Shohibussirri, M., Sichinava, D., Silveira, A., Silveira, N., Simi, M., Simionescu, R., Simkó, K., Šimková, M., Simov, K., Skachedubova, M., Smith, A., Soares-Bastos, I., Spadine, C., Stella, A., Straka, M., Strnadová, J., Suhr, A., Sulubacak, U., Suzuki, S., Szántó, Z., Taji, D., Takahashi, Y., Tamburini, F., Tanaka, T., Tella, S., Tellier, I., Thomas, G., Torga, L., Toska, M., Trosterud, T., Trukhina, A., Tsarfaty, R., Türk, U., Tyers, F., Uematsu, S., Untilov, R., Urešová, Z., Uria, L., Uszkoreit, H., Utka, A., Vajjala, S., van Niekerk, D., van Noord, G., Varga, V., Villemonte de la Clergerie, E., Vincze, V., Wakasa, A., Wallin, L., Walsh, A., Wang, J. X., Washington, J. N., Wendt, M., Widmer, P., Williams, S., Wirén, M., Wittern, C., Woldemariam, T., Wong, T.-s., Wróblewska, A., Yako, M., Yamashita, K., Yamazaki, N., Yan, C., Yasuoka, K., Yavrumyan, M. M., Yu, Z., Žabokrtský, Z., Zeldes, A., Zhu, H., and Zhuravleva, A. (2020). Universal Dependencies 2.6. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University. Cited on page 18.

Zeman, D., Popel, M., Straka, M., Hajic, J., Nivre, J., Ginter, F., Luotolahti, J., Pyysalo, S., Petrov, S., Potthast, M., Tyers, F., Badmaeva, E., Gokirmak, M., Nedoluzhko, A., Cinkova, S., Hajic jr., J., Hlavacova, J., Kettnerová, V., Uresova, Z., Kanerva, J., Ojala, S., Missilä, A., Manning, C. D., Schuster, S., Reddy, S., Taji, D., Habash, N., Leung, H., de Marneffe, M.-C., Sanguinetti, M., Simi, M., Kanayama, H., dePaiva, V., Droganova,

K., Martínez Alonso, H., Çöltekin, Ç., Sulubacak, U., Uszkoreit, H., Macketanz, V., Burchardt, A., Harris, K., Marheinecke, K., Rehm, G., Kayadelen, T., Attia, M., Elkahky, A., Yu, Z., Pitler, E., Lertpradit, S., Mandl, M., Kirchner, J., Alcalde, H. F., Strnadová, J., Banerjee, E., Manurung, R., Stella, A., Shimada, A., Kwak, S., Mendonca, G., Lando, T., Nitisaroj, R., and Li, J. (2017). CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–19. Association for Computational Linguistics. Cited on pages 51, 69, 70, 83, 103, 108, and 122.

Zeman, D. and Resnik, P. (2008). Cross-Language Parser Adaptation between Related Languages. In *Proceedings of the IJCNLP-08 Workshop on NLP for Less Privileged Languages*. Cited on pages 5, 73, 102, 103, 104, 105, and 106.

Zeman, D. and Žabokrtský, Z. (2005). Improving Parsing Accuracy by Combining Diverse Dependency Parsers. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 171–178, Vancouver, British Columbia. Association for Computational Linguistics. Cited on pages 63, 64, 65, and 67.

Zhang, H. and McDonald, R. (2012). Generalized Higher-Order Dependency Parsing with Cube Pruning. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 320–331, Jeju Island, Korea. Association for Computational Linguistics. Cited on page 40.

Zhang, M., Zhang, Y., Che, W., and Liu, T. (2014). Character-Level Chinese Dependency Parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1326–1336, Baltimore, Maryland. Association for Computational Linguistics. http://www.aclweb.org/anthology/P14-1125. Cited on page 20.

Zhang, Y. and Barzilay, R. (2015). Hierarchical Low-Rank Tensors for Multilingual Transfer Parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1857–1867, Lisbon, Portugal. Association for Computational Linguistics. Cited on page 102.

Zhang, Y. and Clark, S. (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*, pages 562–571, Honolulu, Hawaii, USA. Association for Computational Linguistics. Cited on pages 23, 56, 66, and 85.

Zhang, Y. and Nivre, J. (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*, pages 188–193, Portland, Oregon, USA. Association for Computational Linguistics. Cited on pages xiii, 29, 30, and 85.

Zhang, Y. and Weiss, D. (2016). Stack-propagation: Improved Representation Learning for Syntax. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1557–1566, Berlin, Germany. Association for Computational Linguistics. Cited on page 127.

Zhang, Z. and Zhao, H. (2015). High-order Graph-based Neural Dependency Parsing. In *Proceedings of the 29th Pacific Asia Conference on Language, Information and Computation*, pages 114–123. Cited on pages ix, 32, and 143.

Zhou, Z. (2007). Entwicklung einer französischen Finite-State-Morphologie. Diplomarbeit, Institute for Natural Language Processing, University of Stuttgart. Cited on page 84.

Zhou, Z.-H. (2012). *Ensemble Methods: Foundations and Algorithms*. Chapman and Hall/CRC, New York. Cited on page 60.

Zhou, Z.-H. and Li, M. (2005). Tri-training: Exploiting unlabeled data using three classifiers. *IEEE Transactions on knowledge and Data Engineering*, 17(11):1529–1541. Cited on page 66.

Zsibrita, J., Vincze, V., and Farkas, R. (2013). Magyarlanc 2.0: Szintaktikai elemzés és felgyorsított szófaji egyértelműsítés. In *IX. Magyar Számítógépes Nyelvészeti Konferencia*, pages 368–376. Cited on page 84.

Typeset July 1, 2021