ACTA UNIVERSITATIS UPSALIENSIS
Studia Linguistica Upsaliensia
24

Linguistically Informed Neural Dependency Parsing for Typologically Diverse Languages

Miryam de Lhoneux

UPPSALA
UNIVERSITET

Dissertation presented at Uppsala University to be publicly examined in Bertil Hammer, Blåsenhus, von Kraemers Allé 1, Uppsala, Monday, 25 November 2019 at 13:15 for the degree of Doctor of Philosophy. The examination will be conducted in English. Faculty examiner: Professor Emily Bender (University of Washington, Department of Linguistics).

Abstract

de Lhoneux, M. 2019. Linguistically Informed Neural Dependency Parsing for Typologically Diverse Languages. *Studia Linguistica Upsaliensia* 24. 178 pp. Uppsala: Acta Universitatis Upsaliensis. ISBN 978-91-513-0767-1.

This thesis presents several studies in neural dependency parsing for typologically diverse languages, using treebanks from Universal Dependencies (UD). The focus is on informing models with linguistic knowledge. We first extend a parser to work well on typologically diverse languages, including morphologically complex languages and languages whose treebanks have a high ratio of non-projective sentences, a notorious difficulty in dependency parsing. We propose a general methodology where we sample a representative subset of UD treebanks for parser development and evaluation. Our parser uses recurrent neural networks which construct information sequentially, and we study the incorporation of a recursive neural network layer in our parser. This follows the intuition that language is hierarchical. This layer turns out to be superfluous in our parser and we study its interaction with other parts of the network. We subsequently study transitivity and agreement information learned by our parser for auxiliary verb constructions (AVCs). We suggest that a parser should learn similar information about AVCs as it learns for finite main verbs. This is motivated by work in theoretical dependency grammar. Our parser learns different information about these two if we do not augment it with a recursive layer, but similar information if we do, indicating that there may be benefits from using that layer and we may not yet have found the best way to incorporate it in our parser. We finally investigate polyglot parsing. Training one model for multiple related languages leads to substantial improvements in parsing accuracy over a monolingual baseline. We also study different parameter sharing strategies for related and unrelated languages. Sharing parameters that partially abstract away from word order appears to be beneficial in both cases but sharing parameters that represent words and characters is more beneficial for related than unrelated languages.

Keywords: Dependency parsing, multilingual NLP, Universal Dependencies, Linguistically informed NLP

Miryam de Lhoneux, Department of Linguistics and Philology, Box 635, Uppsala University, SE-75126 Uppsala, Sweden.

ISSN 1652-1366
ISBN 978-91-513-0767-1
urn:nbn:se:uu:diva-394133 (http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-394133)

Contents

List of Tables

List of Figures

Nomenclature

AVC	Auxiliary Verb Construction
BiLSTM	Bidirectional Long Short Term Memory Network
FMV	finite main verb
K&G	Parsing architecture from Kiperwasser and Goldberg (2016b)
LAS	Labeled Attachment Score
LSTM	Long Short-Term Memory Network
MAUX	main auxiliary
MLP	Multi-layer perceptron
MTL	Multi-task learning
NFMV	non-finite main verb
NLP	Natural Language Processing
POS	part-of-speech
RNN	Recurrent Neural Network
S-LSTM	Stack LSTM parser (Dyer et al., 2015)
SVM	Support Vector Machine
UAS	Unlabeled Attachment Score
UD	Universal Dependencies

Acknowledgments

First and foremost, I would like to thank Joakim Nivre. Your research has been inspiring me from before my PhD and you have been an exceptional advisor. You provided a *lagom* amount of guidance and freedom. Your thoroughness has been frustrating at times but has certainly made me a better researcher. Thanks to my second supervisor Sara Stymne, you jumped on my PhD train in my second year and stayed on board even through your parental leave. *Tack* both of you for your constructive comments and all our discussions! This PhD experience has been mostly enjoyable, I was told I looked surprisingly relaxed (especially during the write-up) and I have the two of you to thank for that!

I was also lucky to work here with Aaron Smith, who really improved the code of the parser, but also with Artur Kulmizev (who is also a great office-mate!), Ali Basirat and Yan Shao. The parsing meetings which have included Joakim, Sara, Aaron, Artur, Ali but also Johannes Gontrum and group visitors, have been a great source of inspiration and the beginning of several papers.

My other colleagues in the Computational Linguistics group have made this place a nice working environment! These include Fabienne Cap (my work wife for my first 2 years, who was also great fun teaching with), Marie Dubremetz (*ça chiasme pour elle*), Gongbo Tang (food master), Sharid Loáiciga (stupid pronouns!), Christian Hardmeier, Mats Dahllöf, Eva Pettersson, Beáta Megyesi and Fredrik Wahlberg. Thanks also to all my students in the NLP course!

The working environment at the department has been joyful and stimulating thanks to these direct colleagues but also thanks to Marc Tang (Yahaa!), Rima Haddad (organiser of everything fun including the choir), Josefin Lindgren (a great officemate!), Linnéa Öberg (read that one singing), Buket Öztekin, Karin Koltay (our choir conductor), Klara Bertils (our choir conductor too), Maja Robbers, Philipp Rönchen, Harald Hammarström, and others. Thanks to those who were in LingSing and those who came for Friday beers at Williams! Thanks to Aaron and (Comed)Yan for the guitar sessions!

We were lucky to have many visitors here who were all interesting to talk to and fun to hang out with. Discussions I had with Yoav Goldberg shaped some of the ideas of this thesis, and I worked with Eli Kiperwasser on the parser for the shared task (aka *#ParsingTragedy*). It was fun coding side by side. Other notable visitors include Liane, Prasanth, Amir, Zhengxian, Gita and Paola(s).

I was myself very lucky to visit Copenhagen for 2 months where I learned a lot from working with Anders Søgaard, Johannes Bjerva and Isabelle Augenstein in a very collaborative environment. Thanks to the rest of the group who made this experience a fun adventure: Joachim, Mareike, Yova, Marcel, Maria. I was also lucky to intern at IBM in New York with Miguel Ballesteros which was an incredible experience. *Gracias* Miguel for being such a good mentor!

I want to thank Marco Kuhlmann for being opponent for my final seminar. He provided thorough comments, down to the smallest latex details, and we had a great discussion which helped improve this thesis. Thanks James Barry for proofreading an early draft. Thanks Rima Haddad for designing the cover figure, inspired by the SyntaxFest logo by Kim Gerdes.

I acknowledge the computational resources provided by CSC in Helsinki and Sigma2 in Oslo through NeIC-NLPL (www.nlpl.eu) which have been used for all experiments of this thesis. Thanks to all the UD contributors for making this thesis possible. Thanks Sweden for being the best country for a PhD.

I want to thank the *#NLProc* twitter community which has had a big impact on my PhD experience. It has made me feel welcome in the community and allowed me to discuss NLP-related things with many people across the globe, some of whom I later met at conferences. Thanks to my conference and summer school (the cool kids!) buddies, whether twitterati or not, for all the discussions which have helped this thesis in one way or another. Thanks to Simon Hengchen for support in being a fellow Belgian NLPer in the north.

My PhD is the result of a long journey. I cannot relate the start of my passion for language to any event in particular but I remember vividly that my passion for syntax started in the classroom of Joachim Sabel back in Belgium. My cousin Louis de Viron may have pushed me towards the computational side of linguistics faster than I would otherwise have done, by enthusiastically talking about NLP. Many lecturers in Belgium and Edinburgh have contributed to shaping me as a computational linguist and as a researcher. In particular, working with my linguistics master's thesis supervisor, Gaëtanelle Gilquin, made me better at communicating about science. I learned a lot about experimental design from Omri Abend and about linguistics in NLP from Mark Steedman, my NLP master's thesis supervisors. I also learned a lot about AI from working with Gerben van Eerten at Crowdynews before my PhD.

My time in Uppsala has been great, thanks in no small part to meetup organisers who have made it so easy to build a social network and have now become my friends. These include Tabea, Bethany, Ravi, Tobias, Nikos, Masoud, Emma, Cai, etc. Thanks to my badminton friends for helping me let the steam out and thanks to my beer, hike and other adventures friends. Thanks also to my Belgian crowd for bringing home to Uppsala every once in a while.

I want to thank my friends in Belgium and other places (especially my fellow Edinburgh alumni, aka the AT gang) for still being there despite the distance.

Importantly, I want to thank my family for being supportive and fun: papa et mamy, Armand et Johanna, Valérie et François, Carmen, et Constantin. My parents always encouraged me to follow my professional goals even if it meant moving norther and norther. They also taught me that the people we do things with matter more than the things we do, which makes me truly value collaborations. *Merci pour tout!* My maternal grandparents saw the beginning of this PhD but not the end. Their love and support still carry me.

Last but not least, Lars: you came into my life halfway in my PhD and quickly became my favourite person. Thank you for all the tasty/healthy dishes and most of all, thank you for the support, it's been incredibly comforting to have someone who's been through this by my side. *Danke*, you're the best!

1. Introduction

The field of Natural Language Processing (NLP) is currently seeing an increase in multilingual resources and this has led to an increase in studies on cross-lingual transfer.[1] Having a multilingual dataset allows two things: 1) we can build models that rely on linguistic knowledge that is neither too general to be useful nor specific to one language, 2) we can leverage data from multiple languages to make our tools better for low-resource languages.

Dependency parsing, the task of assigning a syntactic structure to a sentence, is central to NLP and useful to many applications. Recent examples of using dependency parsing for an NLP application include Chen et al. (2017) for machine translation, Tai et al. (2015) for sentiment analysis, Ma et al. (2016) for coreference resolution and Strubell et al. (2018) for semantic role labelling.

It is a task for which we have had a multilingual dataset for a long time: treebanks in 19 languages were released in the context of the CoNLL 2006 and 2007 shared tasks (Buchholz and Marsi, 2006; Nivre et al., 2007). A limitation with these treebanks has been that they are annotated with different schemes which makes it difficult to take advantage of these resources, makes it difficult to evaluate and compare results on different treebanks, and has led to limited gains from cross-lingual training. These shortcomings are being addressed with the development of Universal Dependencies (UD), a recent project that is seeking to harmonise the annotation of treebanks across languages (Nivre et al., 2016a). UD has released an increasingly large number of treebanks in the last few years with a large variety of languages. In this thesis, we first exploit this new resource to build and analyse models *for typologically diverse languages*. We then attempt to make our models *linguistically informed*: we seek to incorporate linguistic knowledge into our models as well as characterise what they learn about language. Finally, having models that work well for typologically diverse languages allows us to build better cross-lingual models.

Dependency parsing, like many other tasks in NLP, has seen a large increase in accuracy with the development of *neural* methods. Neural models have been largely used as black boxes, but recent work has started investigating how linguistic knowledge can be incorporated into these models and what they learn about language. This thesis presents work in these two directions with application to dependency parsing. We investigate *linguistically informed neural dependency parsing for typologically diverse languages*.

[1]Plank (2019) reports a large increase in papers including the word *cross-lingual* in the title in the ACL anthology over the last 15 years, with no more than 7 papers per year from 2004 to 2008 and a record number of 81 papers in 2019.

1.1 Research Questions

At a high level of abstraction, the overarching question I ask in this thesis is the following:

RQ0 How can linguistics inform neural NLP?

I ask this question in the context of neural dependency parsing for typologically diverse languages. Dependency parsing that works for typologically diverse languages using UD comes with challenges which we first need to address. The first main research question we therefore address is the following:

RQ1 How can we build parsing models that perform well across typologically diverse languages?

One challenge is that UD treebanks contain a biased selection of languages, with a majority of Indo-European languages. We need to take care to build models that are not biased towards language families that are overrepresented.

Another challenge when dealing with a varied set of languages is morphologically complex languages which have historically been harder to parse than languages that encode syntactic structure mainly through word order and function words. A last challenge is non-projectivity, a property of certain dependency trees which is notoriously difficult to handle. It can be ignored when working with languages such as English, because non-projectivity is typically a rare phenomenon in English treebanks. It cannot be ignored without serious costs in parsing accuracy when working with languages that have treebanks where this phenomenon is frequent like Ancient Greek.

We attempt to address these challenges. Having done that, we can subsequently investigate how to make our models linguistically informed by studying the incorporation of linguistic knowledge into our model as well as investigate what these models learn about language. We are now also equipped with a model suited for cross-lingual transfer. I identify three potential areas of research in order to work in this direction: incorporating linguistically informed inductive biases in neural architectures, interpreting what neural models learn about language, and making use of multi-task learning. This allows me to formulate the three remaining main research questions. The second main research question of this thesis is the following:

RQ2 How can we incorporate linguistic knowledge into neural models for dependency parsing?

Among studies that seek to incorporate inductive biases into neural architectures, several investigate the incorporation of a hierarchical bias. Dyer et al. (2015) argue that language is hierarchical and that such knowledge should be incorporated into parsing models. They study the impact of a specific method for incorporating that hierarchical bias. They propose a model which uses both information constructed sequentially (sequential layers) and information constructed hierarchically (a tree layer) and demonstrate the importance of this tree layer. We pursue this line of research.

The third main research question that we look at is the following:

RQ3 How can we investigate what neural models learn about language when trained for the task of dependency parsing?

A method which has become standard in analysing what neural models learn is the method of diagnostic classifiers, introduced independently in various studies: Adi et al. (2017), Ettinger et al. (2016) and Hupkes et al. (2018). We use this method to investigate our parsing model. We focus on what our parser learns about a specific construction: the Auxiliary Verb Construction (AVC). This construction is interesting for different reasons. In UD, it is represented with the main verb as head and the auxiliary as dependent. The choice of this representation is motivated by typological concerns: UD favours an annotation where function words are dependent of content words and not the other way around. This specific choice for AVCs is controversial in that previous research has shown that having the auxiliary as head is better for parsing accuracy. We found in previous work that the UD representation is better than the alternative when using a pre-neural parser. We investigate this question in neural parsing. Another reason for why this construction is interesting is that it represents the phenomenon of *dissociated nucleus*: words that are related which share head properties. In the work of Tesnière (1959), a pioneer of Dependency Grammar, the relation that holds between these two words is not a dependency relation but a relation of *transfer*. UD has chosen a representation of AVCs where this interpretation can be read from the labels. However, this has not been made explicit in parsing and we find it interesting to explore whether current models do learn this notion.

The final research question we look at is the following:

RQ4 How can we leverage information from multiple treebanks in different languages to improve parsing performance on the individual languages?

Some work has shown benefits from training parsers multilingually for high- and low-resource languages. Training a parser simultaneously for multiple languages (polyglot parsers) can lead to improvements in parsing accuracy over training multiple parsers monolingually. We pursue this line of research.

1.2 Outline

I start by giving more background in Chapter 2. I describe UD, its relation to dependency grammar, its development and its design principles. I also describe dependency parsing and parsing with UD, with a focus on the shift towards using neural models. I finally describe work that either incorporates linguistic knowledge into neural models or that characterises what neural models learn about language.

Chapter 3 attempts to address the challenges that come with working with a dataset containing typologically diverse languages described earlier. With this, we seek to provide answers to **RQ1**. We propose to sample a subset of UD treebanks that is representative of the variety of treebanks for parsing development and evaluation. We build on work that has used character models as a proxy for morphological information and show that such models robustly help all languages and are particularly useful for morphologically complex languages. We finally investigate non-projective parsing. The outcomes of this chapter are a parser that works well on UD treebanks as well as a general method to do further parser development. Both are used in the remaining chapters.

Chapter 4 investigates the use of a hierarchical bias in our parser by making use of a tree layer. With this, we seek to provide answers to **RQ2**. We re-evaluate the finding that a hierarchical bias is crucial for parsing by looking at a simpler model than in previous studies, which allows evaluating the contribution of the different parts more easily, and by looking at a larger variety of languages. We cast doubt on this finding and further investigate the interaction between the tree layer and other parts of our model, thereby shedding light on the conditions under which this tree layer is useful in neural parsing.

In Chapter 5, we attempt to characterise what neural models learn about language when trained for the task of parsing, with a focus on auxiliary verb constructions. With this, we seek to provide answers to **RQ3**. We first verify the finding that the UD representation of AVCs is better for parsing than the alternative with our neural parser. Then we look at agreement and transitivity information learned about AVCs by the parser, to investigate the question of whether or not the parser learns the notion of dissociated nucleus. The tree layer we explore prior to that is particularly relevant here and we look at whether a tree layer can help learn this notion.

In Chapter 6, we explore polyglot parsing. We obtain large improvements from training a polyglot parser for related languages over training multiple monolingual parsers. We then investigate different parameter sharing strategies for related and unrelated languages in a restricted setting where we train parsers on language pairs, related or unrelated. With this, we seek to provide answers to **RQ4**.

I conclude the thesis, including providing some answers to **RQ0**, and give perspectives for future work in Chapter 7.

1.3 Relation to Published Work

The studies presented in this thesis have been partially published in the following papers:

- Miryam de Lhoneux and Joakim Nivre. 2016. Should Have, Would Have, Could Have. Investigating Verb Group Representations for Parsing with Universal Dependencies. In *Proceedings of the Workshop on Multilingual and Cross-lingual Methods in NLP*, pages 10–19.

Parts of the methodology from Chapter 5, Section 5.2 were introduced in that paper. This methodology was developed in collaboration with my co-author.

- Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2017c. Old School vs. New School: Comparing Transition-Based Parsers with and without Neural Network Enhancement. In *Proceedings of the 15th Treebanks and Linguistic Theories Workshop (TLT)*, pages 99–110.

This paper introduced the idea of sampling and the criteria presented in Chapter 3, Section 3.1. All ideas came from discussions with my co-authors.

- Miryam de Lhoneux, Yan Shao, Ali Basirat, Eliyahu Kiperwasser, Sara Stymne, Yoav Goldberg, and Joakim Nivre. 2017a. 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.

This is the Uppsala system paper for the CoNLL 2017 shared task. The results are presented in Chapter 3, Section 3.4. I contributed most of the parsing experiments, with the exception of multilingual models. I was the main developer of the parser, including making it possible to build multilingual models, with some help from Eli Kiperwasser and Sara Stymne.

- Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2017b. Arc-Hybrid Non-Projective Dependency Parsing with a Static-Dynamic Oracle. In *Proceedings of the 15th International Conference on Parsing Technologies*, pages 99–104.

This paper introduces the Static-Dynamic oracle presented in Chapter 3, Section 3.3, which adds more extensive background. I contributed the implementation and all experiments. All ideas came from discussions with my co-authors.

- Aaron Smith, Bernd Bohnet, Miryam de Lhoneux, Joakim Nivre, Yan Shao, and Sara Stymne. 2018a. 82 treebanks, 34 models: Universal dependency parsing with multi-treebank models. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 113–123.

This is the Uppsala system paper for the CoNLL 2018 shared task. The results are briefly described in Chapter 3, Section 3.4 and then to a larger extent in Chapter 6, Section 6.2. I contributed with parser development helping Aaron Smith and Sara Stymne who ran most of the parsing experiments. I took part in discussions on the shared task and contributed experiments on using morphological features for parsing which we described in the paper but did not end up using in the system.

- Miryam de Lhoneux, Johannes Bjerva, Isabelle Augenstein, and Anders Søgaard. 2018. Parameter sharing between dependency parsers for related languages. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4992–4997.

This paper explores parameter sharing strategies for polyglot parsing, described in Chapter 6, Section 6.2. I contributed, the implementation and all experiments. All ideas came from discussions with my co-authors.

- Miryam de Lhoneux, Miguel Ballesteros, and Joakim Nivre. 2019a. 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.

This paper investigates recursive composition in our parser, as presented in Chapter 4. I contributed the implementation and all experiments. Ideas came from discussions with co-authors as well as discussions with Sara Stymne and Aaron Smith.

- Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2019b. What Should/-Do/Can LSTMs Learn When Parsing Auxiliary Verb Constructions? *arXiv preprint arXiv:1907.07950*. Under review..

This paper investigates what our parser learns about auxiliary verb constructions, as described in Chapter 5, Section 5.3. I contributed, the implementation and all experiments. All ideas came from discussions with my co-authors.

Throughout the thesis, I use ‘we’ to refer to work that is based on those publications and which were therefore based on collaborative work. I use ‘I’ otherwise.

2. Background

This chapter describes background literature relevant for the thesis. I start by briefly introducing dependency grammar and its use in dependency parsing. After that, I introduce the Universal Dependencies (UD) project and the multilingual treebanks it has released. I then describe dependency parsing and the recent developments in neural parsing leading to the current state-of-the-art models. I finally describe efforts at integrating linguistic information into neural NLP models and, conversely, analysing neural NLP models with respect to linguistic phenomena. This will allow me to refine the research questions described in Chapter 1 and introduce the main studies carried out for this thesis. I describe this in Section 2.5.

2.1 Dependency Grammar

Dependency grammar is a framework for syntactic analysis with diverse theoretical traditions but some common principles. A basic principle is that the syntactic structure of a sentence can be described, at least to a large extent, in terms of asymmetric relations between individual words: dependency relations. For example, in the sentence *I love syntax*, there is a dependency relation between a dependent *I* and head *love* as well as one between a dependent *syntax* and head *love*. The set of dependency relations for the sentence forms a tree, as depicted in Figure 2.1. The (unlabeled) tree can also be illustrated as in Figure 2.2. There is some disagreement on the notion of headedness but typically, the choice of a head in a dependency relation relies on criteria such as whether one of the two words is obligatory (if a word can be dropped, it is a dependent) or whether one word selects or constrains the other.

A dependency tree represents functional relations between words: *I* is the subject of *love* and *syntax* is the object of *love*. This is in contrast with grammar theories where the emphasis is on representing the structure of the sentence such as in phrase structure grammars. In a phrase structure grammar, the syntactic representation of the same sentence would look like in Figure 2.3, where

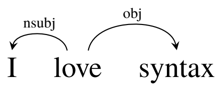


Figure 2.1. A dependency tree.

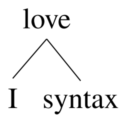


Figure 2.2. Alternative illustration of an (unlabeled) dependency tree.

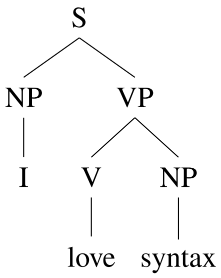


Figure 2.3. A phrase structure tree.

love syntax forms a verb phrase constituent which is combined with the noun phrase *I* to form a sentence S. A dependency tree, unlike a phrase structure tree, is insensitive to word order: the representation of a sentence is the same for two versions of a sentence with a different word order, like for example *Yesterday I went running* and *I went running yesterday*. Both sentences can be represented as in Figure 2.4. This property has made dependency grammar attractive for languages that have flexible word order and some of those languages, for example Slavic languages, have strong traditions in dependency grammar.

The task of dependency parsing, which will be discussed in more detail in Section 2.3, is the task of assigning a dependency tree to a sentence. For practical purposes, dependency parsing has been working with dependency trees where not just *most* but *all* relations that hold between words in the sentence are asymmetric dependency relations, and where those dependency relations form a single tree. With that definition, words are the basic unit of sentences. This is a simplification compared to most dependency grammar theories where representations are considerably more complex, often consisting of multiple strata as in Meaning-Text Theory (Mel'čuk, 1988) and Functional Generative Description (Sgall et al., 1986). In the seminal work of Tesnière (1959), a single level of representation is used, but the basic unit of syntax is not the word but the more abstract notion of *nucleus*. Nuclei often correspond to individual words but sometimes correspond to several words, typically a content word

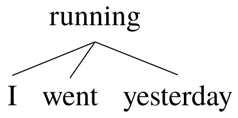


Figure 2.4. Tree for *I went running yesterday* and *Yesterday I went running*.

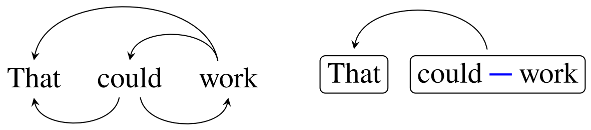


Figure 2.5. Representation of a sentence with auxiliary as used in dependency parsing (left) vs as can be represented following Tesnière (1959) (right). A thick blue line is a transfer relation and a box represents a nucleus.

together with one or more function words, which are said to constitute a *dissociated nucleus*. The internal elements of a dissociated nucleus are connected by *transfer* relations, while the nuclei themselves are connected by *dependency* relations or (in the case of coordination) by *junction* relations.

As de Marneffe and Nivre (2019) describe, all theories of dependency grammar agree on a core set of syntactic constructions. These are constructions where there is a dependency relation between two words and there is a clear choice of head between the two. There are cases, however, where it is clear that there is a relation between two words, but there is no obvious choice of head. This is the case, for example, of the auxiliary verb construction which will be considered at length in Chapter 5. In auxiliary verb constructions, the auxiliary and the main verb share head properties: inflectional verbal features like agreement, tense, aspect, mood, etc. are typically encoded in the auxiliary whereas lexical features like valency are properties of the main verb. In the dependency parsing literature, a sentence with an auxiliary is usually represented as either the top or the bottom tree in the left part of Figure 2.5, with either the auxiliary or the main verb being dependent on the other. If we follow the ideas from Tesnière (1959), it can be represented as in the right part of Figure 2.5 where the auxiliary and main verb are connected by a transfer relation to form a nucleus. This nucleus is itself connected to the other words/nuclei in the sentence. In this example, the word *That* corresponds to a nucleus and the words *could* and *work* are each part of a dissociated nucleus.

See Nivre (2006) and de Marneffe and Nivre (2019) for a more complete overview of dependency grammar, its use in dependency parsing, and its influence on UD.

2.2 Universal Dependencies

Universal Dependencies (Nivre et al., 2016a) is a project that seeks to harmonise the annotation of dependency treebanks across languages. It has released a large number of treebanks in a variety of languages and language families. At the time of writing, the latest release, version 2.4 (Nivre et al., 2019), contains 146 treebanks in 83 languages and 17 language families (although the majority of languages are Indo-European), plus one sign language, one creole and one code-switching treebank. This resource opens many opportunities for the development of multilingual or typologically aware parsing models. First, it allows us to incorporate linguistic knowledge into our models without tying that knowledge to a specific language. This makes it possible to build models that generalise well. Second, it allows us to leverage resources from multiple languages by multilingual training and improve accuracy, especially on the low-resource ones. Many of the UD treebanks are indeed very small: for 19 of the 83 languages in v2.4, there is a maximum of 10k tokens in the language's treebanks. This is orders of magnitude less than the high-resource languages like Czech which has 2,222k tokens in total in its treebanks. The development of UD is the result of several developments in dependency parsing which I set out to describe. I will subsequently describe the design principles of UD.

2.2.1 Historical Background

Dependency grammar is an attractive framework when working in a multilingual perspective since, as described in Section 2.1, dependency trees are not sensitive to the order of the words in a sentence, which allows us to deal with languages with free word order as easily as languages with strict word order (Nivre, 2006; de Marneffe and Nivre, 2019). This is in contrast with phrase structure where the tree structure incorporates word order. Dependency parsing has been grounded in multilingualism from the beginning of its history, driven by the CoNLL 2006 and 2007 shared tasks (Buchholz and Marsi, 2006; Nivre et al., 2007) which released dependency treebanks in 19 languages in total.[1] This focus on multilinguality can even be said to date back to Tesnière (1959) who provided examples of his dependency scheme for dozens of languages, and to Schubert (1987) who describes dependency grammar in a contrastive perspective, with the aim to translate from one language to another.

The dependency treebanks in the 2006 and 2007 CoNLL shared tasks used a similar annotation format but different annotation schemes. Figure 2.6[2] gives the annotation of a sentence in Swedish, Danish and English according to the guidelines from the Swedish Treebank (Nivre and Megyesi, 2007), the Danish Dependency Treebank (Buch-Kromann, 2003) and Stanford Dependencies (de Marneffe et al., 2006), respectively, where the words correspond directly but where the annotations differ substantially. The development of these treebanks has allowed the development of parsers that can be used for different languages but has had the consequence that 1) the linguistic knowledge that can be built into parsing models is limited and 2) it has limited the gains that can be obtained from cross-lingual training.

As far as 1) is concerned, Nivre (2015) argues that NLP systems are usually at either of two ends of a scale: there are systems which make extensive use

[1] 13 and 10 languages for the 2006 and 2007 tasks respectively, with some overlap.

[2] Example taken from de Marneffe and Nivre (2019).

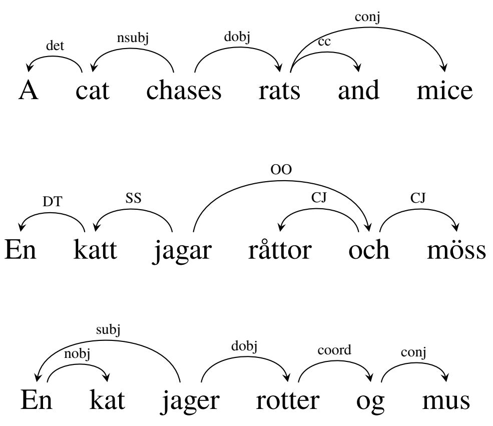

conj
det
nsubj
dobj
cc
A cat chases rats and mice
OO
DT
SS
CJ
CJ
En katt jagar råttor och möss
subj
nobj
dobj
coord
conj
En kat jager rotter og mus

Figure 2.6. Dependency trees of sentences in English, Swedish and Danish in their original annotation scheme.

of knowledge about a language and systems that do not take any characteristic of the language into account. The first type of system is hard to adapt to other languages and the second fails to exploit information that is potentially useful. Having treebanks consistently annotated across languages allows striking a good balance between these two extremes, where we can build knowledge about language variation into our models.

As far as 2) is concerned, McDonald et al. (2011) have shown that delexicalised parsing models trained on a source language, i.e. models that are trained purely on part-of-speech (POS) tags and that do not take word information into account, can successfully be transferred to a target language and outperform an unsupervised model. In that paper, they test this method on all combinations of language pairs for 9 languages (Danish, German, Greek, English, Spanish, Italian, Dutch, Portuguese and Swedish) and find that the usefulness of a source language for its target language is not predictable from language family properties. For example, they find that for Dutch, Greek is the best source language (it leads to the best parsing accuracy) and German the worst. They also find that Danish is the worst source language for Swedish. They conclude that this is due to differences in annotations. This was one of the motivations that led to the creation of UD. McDonald et al. (2013) repeated cross-lingual parser experiments similar to McDonald et al. (2011) but using an early version of UD and they found expected patterns with respect to language family considerations: languages that are closely related improve accuracy on a target language more than languages that are less closely related. They conclude that the gains of cross-lingual transfer have been underestimated by McDonald et al. (2011) and that UD allows better cross-lingual transfer.

Another motivation for the development of UD is the practical purpose of having a common representation for downstream tasks: it makes it possible to use tools which use predicted dependency analysis for the languages that share that representation. de Marneffe et al. (2006) built a dependency scheme for English, the Stanford Dependencies, with practical applications in mind. This scheme was then extended to more languages in de Marneffe et al. (2014) and most of its principles were carried over to UD. UD merges ideas from independent initiatives to harmonise the annotation of syntactic relations (Zeman et al., 2012; de Marneffe et al., 2014), POS tags (Petrov et al., 2012) and morphological information (Zeman, 2008).

A final problem which came with working with dependency treebanks with different standards was that it was impossible to compare the performance of systems on different treebanks. Having a common annotation scheme makes this possible, even though Nivre and Fang (2017) provide evidence that we have not yet found a good metric for system comparisons across languages and there is still room for improvement in that respect. In particular, they find that Labeled Attachment Score (LAS), the most common metric for parsing accuracy (which will be introduced in Section 2.3) is a metric that is biased towards analytic languages where grammatical structure tends to be encoded in function words, whereas it is encoded as inflectional morphemes in morphologically complex languages. They propose an alternative metric which gives more importance to relations between two content words than to relations that involve a function word.

2.2.2 Design Principles

The UD guidelines were defined with the following goals which are listed on the UD homepage:[3]

1. UD needs to be satisfactory on linguistic analysis grounds for individual languages.
2. UD needs to be good for linguistic typology, i.e., providing a suitable basis for bringing out cross-linguistic parallelism across languages and language families.
3. UD must be suitable for rapid, consistent annotation by a human annotator.
4. UD must be suitable for computer parsing with high accuracy.
5. UD must be easily comprehended and used by a non-linguist, whether a language learner or an engineer with prosaic needs for language processing. We refer to this as seeking a habitable design, and it leads us to favor traditional grammar notions and terminology.
6. UD must support well downstream language understanding tasks (relation extraction, reading comprehension, machine translation, …).

[3]See `https://universaldependencies.org/introduction.html`

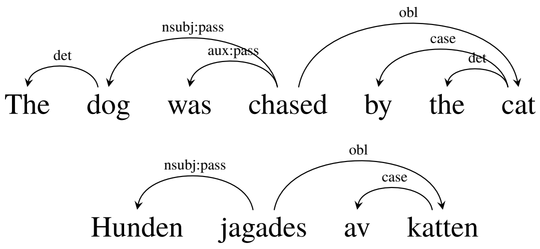


Figure 2.7. Parallel sentences in the passive voice in English and Swedish.

The different goals can of course be in conflict. For example, UD strongly emphasises the primacy of content words for reasons which will be discussed later in this section, and always favours attaching function words to content words rather than the other way around which, some research has shown, can be suboptimal for parsing, as mentioned by de Marneffe et al. (2014). The aim is therefore to find a good compromise between those goals.

UD follows its predecessor, Stanford Dependencies, in adopting the *lexicalist hypothesis* where words are the basic unit of syntactic analysis and syntactic relations hold between words in a sentence. This decision is motivated linguistically and is practical because most NLP tools assume that sentences are tokenized into words. This choice therefore fulfills goals 1, 2, 5 and 6: it is motivated linguistically, in a typological perspective, and is intuitive and useful for downstream tasks.

Another important choice in UD is to have as a fundamental principle that dependency relations hold primarily between content words. This choice was made with goals number 2, 5 and 6 in mind. This principle maximises parallelism across languages (goal 2) because some grammatical functions that tend to be expressed by function words in some languages, tend to be expressed by morphological inflection in others or not expressed at all, like for example tense. An example[4] is provided by the two parallel sentences in English and Swedish in Figure 2.7, where the passive voice is encoded as a function word *was* with a participle in English and as an inflection of the main verb *jagades* in Swedish. The annotation of these two sentences is more parallel when *chased* is the main root word than if *was* were the main root word and *chased* attached to it as is done in most dependency grammars, as I discuss next. Croft et al. (2017) provide further arguments that this principle fills goal number 2 by relating it to the typological literature.

A result of this principle, as we can see in the English example, is that function words attach to content words, as mentioned before, and in this particular case, the auxiliary attaches to the main verb. This is a controversial decision since most dependency grammars treat function words as heads (Osborne and

[4]Taken from the UD website `https://universaldependencies.org/introduction.html`

Maxwell, 2015). However, if we follow the ideas of Tesnière (1959), auxiliary verb constructions, among other constructions involving function and content words, form part of a dissociated nucleus and the relation that holds between the main verb and the auxiliary is a relation of transfer, not of dependency. Choosing a head is therefore just a matter of convenience: it allows us to keep this representation as part of a tree, and it does not matter which one we choose. As argued by de Marneffe and Nivre (2019), if we read the label *aux* as indicating a relation of transfer rather than a relation of dependence, we can interpret the tree representation of auxiliary verb constructions as a dissociated nucleus. They argue more generally that UD trees can be interpreted as in the work of Tesnière (1959).

UD treebanks are used in experiments throughout the thesis. We use different versions depending on when the experiments were carried out. Treebank naming conventions have changed between version 2.1 and 2.2: from 2.2, all treebanks are named after the language plus a suffix. Not all treebanks from versions prior to 2.2 have a suffix. Those treebanks usually correspond to the first treebank released for that language and are treated as *default* treebanks for their language. In this thesis, I specify the suffix unless we are using this 'default' treebank. ISO codes based on the treebank's language are often used in tables and figures to save space. A table of languages and their ISO code adopted by UD can be found in Appendix A.

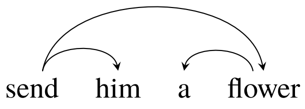
2.3 Dependency Parsing

This section first gives an overview of dependency parsing, including pre-neural methods, and then describes recent developments in neural dependency parsing, leading to the current state of the art. After that, I describe work that seeks to to leverage data from different languages to improve parsing accuracy on low-resource languages.

2.3.1 Dependency Parsing

Dependency parsing is the task of assigning a dependency tree to a sentence. Formally, a dependency tree G is a tuple (V,A) containing a set of vertices V, the nodes of a tree, and a set of arcs A. An arc is a triple (h,r,d) representing a head h, a dependency relation r and a dependent d. For a sentence of n words $w_{1:n}$,[5] the task of dependency parsing is the task of finding a set of dependency arcs A such that A forms a single-rooted, acyclic, directed tree spanning the words in V, the words of the sentence. For example, the set of vertices V in the tree in Figure 2.1 is $\{I, love, syntax\}$ and the set of arcs A is $\{(love, nsubj, I), (love, obj, syntax)\}$. In practice, the indices of words are

[5]I use the notation $x_{1:n}$ to denote a sequence of items $x_1, ..., x_n$.



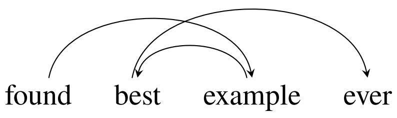


Figure 2.8. A projective and a non-projective tree.

used, so that they are uniquely identifiable. I use indices in the formal descriptions of transition-based systems that follow.

An important property of dependency trees is projectivity. An arc in a tree is projective if there is a directed path from the head word to all the words between the two endpoints of the arc (Kübler et al., 2009), otherwise it is non-projective. More formally, as described in Kübler et al. (2009):

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

where $\rightarrow^*$ means directed path. A projective tree is a tree for which all arcs in A are projective. Figure 2.8 gives an example of a projective and a non-projective tree.[6] The arc between *best* and *ever* is non-projective because there is no directed path from *best* to *example* and *example* is between *best* and *ever*. Non-projective trees are harder to deal with than projective trees. I describe methods to deal with non-projectivity and their difficulties in Section 3.3.

The most common evaluation metrics for dependency parsing are Unlabeled and Labeled Attachment Score (UAS and LAS) where UAS is the percentage of words that have been assigned the correct head and LAS is the percentage of words that have been assigned the correct head and the correct label. Suppose that we predict the tree $\{(love, nsubj, I), (love, nmod, syntax)\}$ for our sentence *I love syntax*, we would have a UAS of 100%, since both heads are correct, and an LAS of 50%, since only one label is correct. As mentioned in Section 2.2.1, LAS might not be the best metric for cross-lingual comparisons and Nivre and Fang (2017) propose an alternative metric which gives more importance to content words, which they call CLAS. This metric is, however, not currently established, and Zeman et al. (2017) report that in the CoNLL shared task 2017, CLAS scores were lower but the system ranking was the same for the two metrics. For these reasons, we exclusively report LAS scores in this thesis.

[6]Thanks to Carlos Gómez-Rodríguez for the non-projective example: `https://twitter.com/carlosgr_nlp/status/908395521081053187`

Initialisation: $c_0(x = (w_1, \ldots, w_n)) = ([\,], [1, \ldots, n, 0], \emptyset)$

Termination: $C_t = \{c \in C \mid c = ([\,], [0], A)\}$

Transition		**Condition**
LEFT-ARC$_d$	$(\sigma\vert i, j\vert\beta, A) \Rightarrow (\sigma, j\vert\beta, A \cup \{(j, r, i)\})$	
RIGHT-ARC$_d$	$(\sigma\vert i\vert j, \beta, A) \Rightarrow (\sigma\vert i, \beta, A \cup \{(i, r, j)\})$	
SHIFT	$(\sigma, i\vert\beta, A) \Rightarrow (\sigma\vert i, \beta, A)$	$i \neq 0$

Figure 2.9. Transitions for the arc-hybrid transition system with an artificial root node (0) at the end of the sentence. The stack Σ is represented as a list with its head to the right (and tail σ) and the buffer B as a list with its head to the left (and tail β).

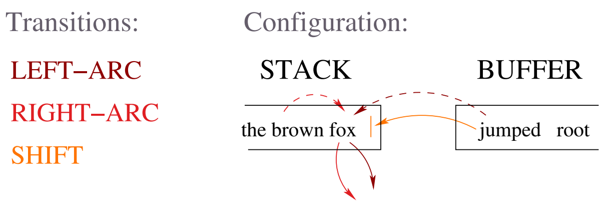


Figure 2.10. Arc-hybrid transitions. A dashed arrow represents an arc and a solid arrow represents movement of an element from the stack or buffer.

There are two main traditions in data-driven dependency parsing: transition-based parsing (Yamada and Matsumoto, 2003; Nivre, 2003), where parsing is treated as a sequence of actions that incrementally connect the words to form a tree, and graph-based parsing (Eisner, 1996; McDonald et al., 2005), where parsing is treated as scoring full trees by decomposing the score into the score of subgraphs. We focus on transition-based parsing in this thesis and therefore describe developments with this method in more detail than developments in graph-based parsing.

Transition-based parsing consists of taking transitions from an initial configuration to a terminal configuration. A configuration typically consists of a stack, a buffer and a set of dependency arcs. The stack contains partially processed words, the buffer contains the remaining words to be processed, and the set of dependency arcs contains the relations that have been added. There are usually up to 4 possible transitions which manipulate the words in the stack and buffer and add dependency arcs. In this thesis, we focus on the arc-hybrid system introduced by Kuhlmann et al. (2011) for reasons described in Chapter 3. I therefore describe it here as an example.

With arc-hybrid, in the initial configuration, indices representing all words from the sentence plus a root node are in the buffer and the stack and the arc set are empty. A terminal configuration has a buffer B with just the root and an empty stack Σ, and the arc set then forms a tree spanning the input sentence.

Table 2.1. *Typical feature templates. s_i and b_i denote the i_{th} word of the stack and buffer respectively. t denotes the POS tag, w denotes the word form, $lc_1(s_1)$ and $rc_1(s_1)$ denote the leftmost and rightmost children of s_1. The symbol $\circ$ denotes feature combination. Table copied from Chen and Manning (2014).*

Single word features (9)
$s_1.w; s_1.t; s_1.wt; s_2.w; s_2.t;$ $s_2.wt; b_1.w; b_1.t; b_1.wt;$
Word–pair features (8)
$s_1.wt \circ s_2.wt; s_1.wt \circ s_2.w; s_1.wt \circ s_2.t$ $s_1.w \circ s_2.wt; s_1.t \circ s_2.wt; s_1.w \circ s_2.w$ $s_1.t \circ s_2.t; s_1.t \circ b_1.t$
Three–word features (8)
$s_2.t \circ s_1.t \circ b_1.t; s_2.t \circ s_1.t \circ lc_1(s_1).t;$ $s_2.t \circ s_1.t \circ rc_1(s_1).t; s_2.t \circ s_1.t \circ lc_1(s_2).t;$ $s_2.t \circ s_1.t \circ rc_1(s_2).t; s_2.t \circ s_1.w \circ rc_1(s_2).t;$ $s_2.t \circ s_1.w \circ lc_1(s_1).t; s_2.t \circ s_1.w \circ b_1.t$

There are three types of transitions, SHIFT, LEFT-ARC$_d$ and RIGHT-ARC$_d$, defined as in Figure 2.9 and illustrated in Figure 2.10. The LEFT-ARC$_d$ transition removes the first item on top of the stack (i) and attaches it as a modifier to the first item of the buffer j with label r, adding the arc (j, r, i). The RIGHT-ARC$_d$ transition removes the first item on top of the stack (j) and attaches it as a modifier to the next item on the stack (i), adding the arc (i, r, j). The SHIFT transition moves the first item of the buffer i to the stack.

A configuration c is represented by a feature function $\phi(\cdot)$ over a subset of its elements and for each configuration, transitions are scored by a classifier. The feature function returns a vector representation of the configuration. In pre-neural dependency parsing such as in MaltParser (Nivre et al., 2006), a prototypical pre-neural parser, this vector is constructed with properties of the configuration such as word form, POS tag or dependency relation of relevant words in the configuration, e.g. the top item of the stack. Features are also typically built from partially constructed trees, for example leftmost or rightmost dependent of a word. These are commonly called *history-based* features. Features involving words not yet processed (items from the buffer), by contrast, are commonly called *lookahead* features. Finally, features can be combined, for example, the POS tag of the top item of the stack and the first item of the buffer. This results in high-dimensional sparse vectors. Table 2.1 presents a typical set of feature templates which is based on a selection of feature templates from Huang et al. (2009) and Zhang and Nivre (2011).

In pre-neural parsing, the classifier is typically a linear classifier, for example a perceptron (Carreras et al., 2006) or a memory-based learning classifier (Nivre, 2003). Alternatively, the classifier can make use of non-linear kernel

Algorithm 1 Transition-based parser training algorithm

```
1:  INIT(w)
2:  for $i \leftarrow$ ITERATIONS do
3:      for sentence $s$ with gold tree $T$ in corpus do
4:          $c \leftarrow c_s(s)$
5:          while $c$ is not terminal do
6:              $t_p \leftarrow \arg\max_t \mathbf{w} \cdot \phi(c,t)$
7:              $t_o \leftarrow o(c,T)$
8:              if $t_p \neq t_o$ then
9:                  UPDATE(w, $\phi(c,t_o), \phi(c,t_p)$)
10:             $c \leftarrow t_o(c)$
11: return w
```

functions such as a support vector machine (SVM), as in Yamada and Matsumoto (2003). The classifier is trained to learn feature weights to score the possible parsing actions. More precisely, training typically involves iterating over the training set sentences, see Algorithm 1. At each step after the initial configuration $c_s(s)$ for the sentence s, we take the highest scoring transition t_p for the current configuration c according to our current weights **w** and the feature function $\phi(c,t)$. If the transition is incorrect, the weights get updated to increase the margin between the correct transition t_o and the prediction t_p (UPDATE($\mathbf{w}, \phi(c,t_o), \phi(c,t_p)$)). The correct transition is determined by an oracle. An oracle is a function $o(c,T)$ which returns the correct transition given a configuration c and the gold tree T.

In practice, there are issues with this training procedure: first, there can sometimes be several correct transitions for a configuration, but the oracle deterministically returns one. This training procedure considers the other correct transitions as incorrect. Second, the parser only encounters configurations that result from taking correct transitions which means that it is unable to recover from errors. At prediction time, this method therefore suffers from error propagation. This training procedure is deterministic and makes use of what is called a static oracle. An improvement over this training procedure is to make use of dynamic oracles instead, as proposed by Goldberg and Nivre (2012) and Goldberg and Nivre (2013). Dynamic oracles define optimal transitions given any configuration. This allows us to explore errors during training. The concept of dynamic oracles will be considered more at length in Chapter 3.

2.3.2 Neural Dependency Parsing

Dependency parsing, like many other NLP tasks, has seen a large increase in accuracy with neural methods. Improvements have been obtained first by replacing the linear classifier with a non-linear classifier, a feed-forward network.

I first describe this augmentation. Second, I describe the construction of dense vectors which replace high dimensional vectors and have led to improvements in parsing speed and accuracy. Lastly, improvements have been obtained from using Recurrent Neural Networks (RNNs) for feature extraction which I describe after that. A popular architecture which uses all of these improvements is the one by Kiperwasser and Goldberg (2016b) and is the one adopted in this thesis. For this reason, I describe it in detail in the last part of this section. I refer to this parsing architecture as K&G in the remainder of this thesis.

Classification with Feed-Forward Networks

A feed-forward network has an input layer x, an output layer y and one or more hidden layers h_i. Feed-forward networks have had success thanks to the use of non-linear activations which allow learning a complex function of the input layer. An example hidden layer is an affine transformation ($Wx+b$) over the input layer x which is passed through a non-linear activation f as in Equation 2.1. The non-linear activation can for example be a *tanh* or a *sigmoid* function. The output layer can for example, in the case of categorical multiclass classification, be a softmax function which returns a probability distribution y over the labels, as in Equation 2.2.

In neural dependency parsing, the classifier used to score transitions is typically replaced by a feed-forward neural network. This allows us to construct a simpler feature function than was done in pre-neural parsing, as the network will learn feature combinations. Titov and Henderson (2007) were the first to show that neural networks allow learning powerful feature combinations for dependency parsing within a complex neural architecture. Attardi et al. (2009) showed the benefit of replacing an SVM classifier with a multi-layer perceptron (MLP) and obtained competitive results in the EVALITA 2009 dependency parsing shared task (Bosco et al., 2009). The method of using a feed-forward network as a classifier was popularised by Chen and Manning (2014) who additionally made use of dense vectors to represent parsing configurations. This makes it possible to construct an overall simpler architecture than in previous work and provides further gains in parsing accuracy. This further change is described next together with their model.

$$h = f(W_1 x + b) \tag{2.1}$$

$$y = softmax(W_2 h) \tag{2.2}$$

Distributed Representations

Chen and Manning (2014), inspired by the success of distributed representations in NLP, proposed to replace high-dimensional sparse vectors with dense vector representations of word forms, POS tags and dependency relations which are learned as part of the network. An advantage of a dense representation of,

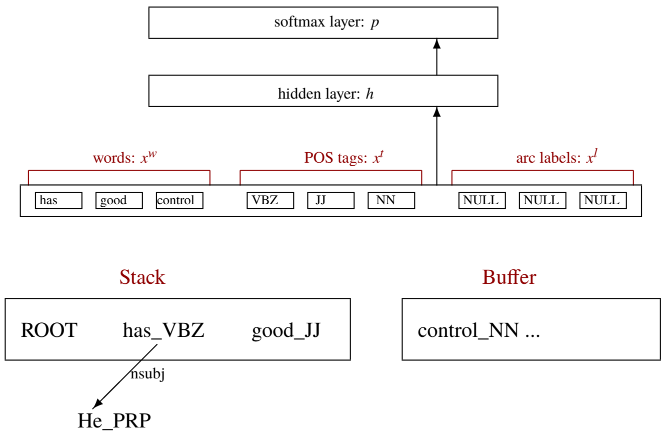


Figure 2.11. Feed-forward neural network parsing architecture from Chen and Manning (2014).

for example, words, besides the compact representation, is that it makes it possible for vectors of similar words to be close to each other in the vector space. As argued by Mikolov et al. (2013a), the notion of similarity between words is not captured in sparse representations.

In order to classify parsing actions, Chen and Manning (2014) select a subset of vectors that are relevant for picking the next action given a configuration. For example, Figure 2.11 represents the network when taking a minimal feature function which includes the word, POS tag and dependency relation of the two top items of the stack and of the first item of the buffer. The dependency relation is undefined for all of these items. Chen and Manning (2014) concatenate the vectors of the words to form x^w (*has*, *good*, and *control* in this case). They concatenate the vectors of the POS to form x^t (*VBZ*, *JJ* and *NN* in this case) and the vectors of the dependency arcs to form x^l (NULL, NULL, NULL). The vectors x^w, x^t and x^l are then passed through a hidden layer h which consists of an affine transformation and a non-linear activation, in this case a cubic activation function as in Equation 2.3. The output of the hidden layer h is then passed through a softmax function which outputs a probability distribution p over the possible transitions as in Equation 2.4. Chen and Manning (2014) use a less minimal feature set than just described, they use more elements from the stack and buffer as well as their leftmost/rightmost children. In the configuration represented in Figure 2.11, we would therefore include *He* in the words vector subset, *PRP* in the POS tags vector subset and *nsubj* in the dependency labels vector subset. They obtain significant gains in accuracy and speed compared to a transition-based parser which uses an SVM and a manually constructed feature set on several datasets.

$$h = (W_1^w x^w + W_1^t x^t + W_1^l x^l + b_1)^3 \tag{2.3}$$

$$p = softmax(W_2 h) \tag{2.4}$$

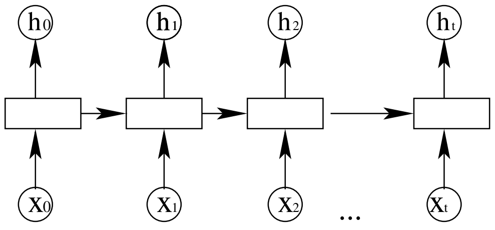


Figure 2.12. RNN architecture. h_t is the output of the RNN for a sequence $x_{1:t}$.

Feature Extraction with RNNs

One of the big successes of neural networks in NLP comes from the use of RNNs (Elman, 1990) which are powerful sequence models.[7] Pre-neural NLP relied extensively on the Markov assumption which says that the future does not depend on the past, given the present. This assumption is necessary to keep models tractable by limiting the features representing a word to a restricted set. RNNs make it possible to construct information about a sequence in an unbounded window and to relax Markov assumptions. An RNN takes a sequence $x_{1:n}$ as input and outputs a representation h of the sequence, as in Equation 2.5. It can also output a summary of the information so far at each step t in the sequence. RNNs can be visualised as a graph as in Figure 2.12 where each hidden cell h_t is a function of the current input cell x_t and the preceding hidden cell h_{t-1}, see Equation 2.6. The value of h_t is therefore defined recursively along the sequence.

$$h = RNN(x_{1:n}) \tag{2.5}$$
$$h_t = f(h_{t-1}, x_t) \tag{2.6}$$

RNNs have known shortcomings: they suffer from vanishing and exploding gradients (Pascanu et al., 2013). As a matter of fact, they perform multiplications recursively which means that, as the sequence gets longer, gradients can diminish and some information gets lost. Conversely, weights in the network can become very large and cause overflow problems. Those shortcomings have been addressed by augmenting the hidden cell h_t with so-called gates which give RNNs the capacity to selectively forget and keep information. An example which is widely used in NLP and will be used in this thesis is the Long Short-Term Memory Network (LSTM), proposed by Hochreiter and Schmidhuber (1997).

[7] See Goldberg (2017) for an extensive description of neural methods in NLP.

The cell of an LSTM is defined as follows:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{2.7}$$
$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{2.8}$$
$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{2.9}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{2.10}$$
$$h_t = o_t \odot tanh(c_t) \tag{2.11}$$

where $\odot$ means element-wise multiplication, the subscript t refers to the time step, σ is the sigmoid function, f_t is the forget gate, i_t is the input gate and o_t is the output gate. The forget gate controls how much of the memory to keep ($f_t \odot c_{t-1}$) and the input gate controls how much of the new information from the current cell should be used ($i_t \odot tanh(W_c x_t + U_c h_{t-1} + b_c)$). This information is stored in the memory cell c_t which is then passed through a non-linearity (a *tanh* here) and controlled by the output gate.

The parser by Chen and Manning (2014) constructs feature vectors that are defined locally, over a subset of words from the stack and buffer. Further improvements can be obtained if we use LSTMs to construct more global information and so that the classifier has access to information about the whole sentence for any given configuration. This was first proposed by Dyer et al. (2015) who constructed a complex parsing architecture which uses LSTMs. This architecture is described more at length in Chapter 4. A simpler but equally effective architecture is the one proposed by Kiperwasser and Goldberg (2016b) which I describe next.

The K&G architecture

Kiperwasser and Goldberg (2016b) use bidirectional LSTMs (BiLSTM) as a feature extractor and keep the rest of the architecture simple. They build both a transition-based and a graph-based parser with this method. We use and extend their transition-based parser in all the remaining chapters of this thesis and therefore describe it here in some detail.

A bidirectional LSTM is an architecture where the sequence $x_{1:n}$ is passed through an LSTM (a *forward* LSTM $\textsc{Lstm}_f$) in its original order and then through another one (a *backward* LSTM $\textsc{Lstm}_b$) in reverse order $x_{n:1}$. The outputs of the forward and the backward LSTM in position i of a word x_i are concatenated to form a representation of the word containing information about the context of the sentence, a *token* representation as in Equation 2.12. I use $[v_1; v_2]$ to denote vector concatenation here and in the remainder of this thesis.

$$\text{BiLSTM}(x_{1:n}, x_i) = [\text{LSTM}_f(x_{1:i}); \text{LSTM}_b(x_{n:i})] \tag{2.12}$$

As described in Section 2.3.1, a configuration c is represented by a feature function $\phi(\cdot)$ over a subset of its elements and for each configuration, transitions are scored by a classifier. In this case, like in Chen and Manning (2014), the classifier is a multi-layer perceptron. The novelty here is that $\phi(\cdot)$ is a concatenation of BiLSTM vectors on top of the stack and the beginning of the buffer, rather than representations of words, POS tags and dependency relations of items of the stack and buffer as well as some of their children. For an input sentence of length n with words $w_{1:n}$, a sequence of vectors $x_{1:n}$ is created, where the vector x_i representing w_i is the concatenation of a word embedding and an embedding of the POS tag t_i of the word w_i.[8] The input vectors x_i are therefore:

$$x_i = [e(w_i); e(t_i)] \tag{2.13}$$

Finally, each input element is represented by a BiLSTM vector, v_i (Equation 2.14). For each configuration c, the feature extractor concatenates the BiLSTM representations of core elements from the stack and buffer: the 3 top items of the stack (s_2, s_1, s_0) and the first item of the buffer (b_0), see Equation 2.15. The MLP scores transitions together with the arc labels for transitions that involve adding an arc (LEFT-ARC$_d$ and RIGHT-ARC$_d$) as in Equation 2.16, where the superscript indicate the different layers. In practice, Kiperwasser and Goldberg (2016b) also use structural features: the leftmost and rightmost dependents of the items of the stack and the buffer which are used as input to the MLP. They find some improvements from using those features. Both the embeddings and the BiLSTMs are trained together with the model. The model is represented in Figure 2.13. The figure and description so far only describe the use of one MLP but in practice, two are used. One MLP scores the transitions, another scores the transition together with the label for the transitions that add a label (RIGHT-ARC$_d$ and LEFT-ARC$_d$). The score is the sum of the scores of the two MLPs.

$$v_i = \text{BiLSTM}(x_{1:n}, i) \tag{2.14}$$

$$\phi(c) = [v_{s_2}; v_{s_1}; v_{s_0}; v_{b_0}] \tag{2.15}$$

$$\text{MLP}(\phi(c)) = W^2 tanh(W^1 \phi(c) + b^1) + b^2 \tag{2.16}$$

This architecture has been widely used in dependency parsing, including in dependency parsing with UD. It was used in 8 of the 10 highest performing systems of the 2017 CoNLL shared task (Zeman et al., 2017). The graph-based version of this architecture has been extended by Dozat and Manning (2017) and that model is currently the state-of-the-art in UD parsing. Dozat et al. (2017) adapted their model to UD by adding representations of characters and obtained the best results in the 2017 CoNLL shared task. That adapted model was used by 9 of the 10 best models in the 2018 CoNLL shared task (Zeman et al., 2018).

[8] In practice, Kiperwasser and Goldberg (2016b) also make use of pretrained word embeddings but this is ignored here for simplicity.

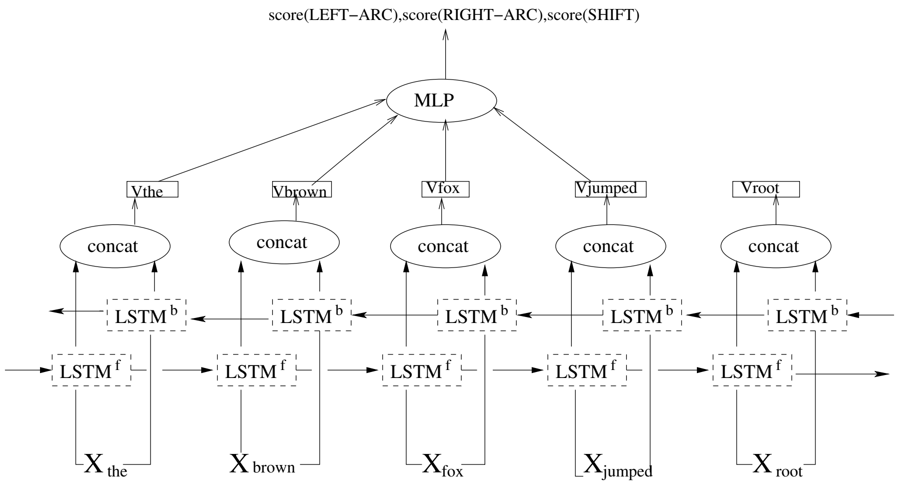


Figure 2.13. Illustration of the neural model scheme of the transition-based parser when calculating the scores of the possible transitions in a given configuration. Here: the stack contains *the brown fox* and the buffer contains *jumped*.

In summary, neural networks have improved dependency parsing in three ways: non-linear classifiers allow learning feature combinations, which has led to improvements over using manually constructed feature sets in accuracy and speed. Secondly, they have allowed replacing high-dimensional vector representations by dense vector representations, which improves parsing speed and accuracy because they are compact and because they make it possible for similar words or POS tags to have similar representations. Lastly, they have allowed constructing feature representations in an unbounded window and take advantage of more global information, i.e. information at the sentence level.

2.3.3 Multilingual Parsing

There is a large body of work on training parsers that are capable of parsing several languages. The terminology is however not always used consistently. In this thesis, I use a terminology depicted in Figure 2.14, inspired in parts by other work.

The largest part of work that attempts to leverage resources from multiple languages does it by training a model on one or several source languages and apply it on a target language. The target language is usually unseen in training or seen but with much less data than the source language(s). This is usually called *cross-lingual* parsing and I use this term in this thesis. A less common alternative is to train a model on several languages with the idea in mind to be able to apply it on all the languages it was trained on. In this setting, all languages have equal status. I will call this method *polyglot parsing* in this thesis. The use of the term *polyglot* to refer to training a model on several languages

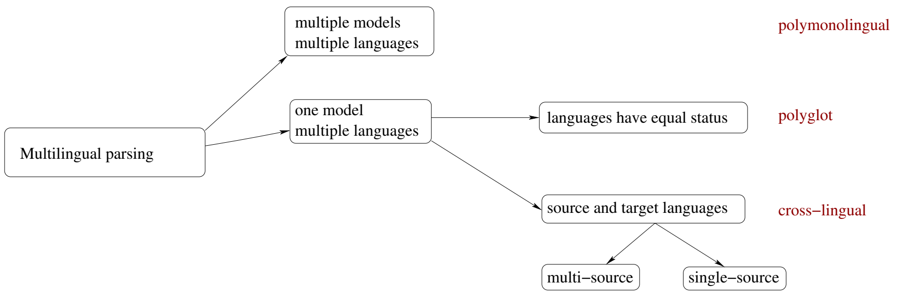

multiple models multiple languages
polymonolingual
one model multiple languages
languages have equal status
polyglot
Multilingual parsing
source and target languages
cross–lingual
multi–source
single–source

Figure 2.14. Multilingual parsing terminology.

with equal status was introduced by Tsvetkov et al. (2016) but is not widely used. Some work, including ours (Smith et al., 2018a), has called these models *multilingual*. However, the term multilingual parsing has also been used in the literature to refer to the fact that the same architecture is used across multiple languages like in Hall et al. (2010). I introduce the term *polymonolingual* for this setting, meaning that we train multiple models monolingually using a common architecture.[9] I use multilingual parsing as a general term that encompasses all work that attempts to work on several languages. Note that this terminology could be applied outside of parsing, in other NLP tasks.

Cross-lingual parsing can be done in many different ways. One type of method is to train a model on a source language and construct bilingual knowledge which helps with transferring the model from the source to the target language. There are multiple ways to build bilingual or multilingual knowledge between the source and target language(s). For example, we can use typological knowledge such as is done in Naseem et al. (2012). Another possibility is to create training data for the target language by projecting the annotation from the source language to the target language using a parallel corpus. This idea was introduced by Yarowsky et al. (2001) for several NLP tasks and applied to parsing among others by Agić et al. (2016). A final possibility is to translate a treebank from a source to a target language and train a parser on the translated treebank, as in Tiedemann et al. (2014). An alternative which does not require constructing bilingual or multilingual knowledge is to train a delexicalised model, usually using POS tags. With this method, a parser is usually trained using POS tag information and can be directly applied to the target language, provided that we can obtain POS tag information for that language. This method was introduced by Zeman and Resnik (2008). For a more comprehensive review of cross-lingual parsing, see Aufrant (2018) who presents an exhaustive typology of the different possible ways to incorporate bilingual or multilingual information which she calls cross-lingual bridges.

The idea of training a polyglot parser has been gaining more attention re-

[9]Thanks to Carlos Gómez-Rodríguez for the suggestion: `https://twitter.com/carlosgr_nlp/status/1133415822427328512`

cently, since having treebanks for different languages annotated with the same guidelines as in UD facilitates this to a great extent. Early examples of trying to exploit this can be found in Vilares et al. (2016) and Ammar et al. (2016). Vilares et al. (2016) trained parsing models on 100 pairs of languages by simply concatenating the treebank pairs. They observed that most models obtained comparable accuracy to the monolingual baseline and some outperformed it. This shows that even in its simplest form, polyglot training can be beneficial. Ammar et al. (2016) build a more complex model to train a parser for eight languages. They make use of additional resources to incorporate typological information (WALS features) as well as cross-lingual information (a dictionary). They additionally modify the architecture to allow the network to capture language-specific information where needed. I describe their approach more in detail in Chapter 6.

Note that a lot of work on cross-lingual or polyglot parsing has been using gold POS tags as part of the input. This is an unrealistic scenario for low-resource languages for which we do not have accurate POS taggers. Even for high-resource languages, this can be a problem. Tiedemann (2015) has shown a large degradation in cross-lingual parsing performance when using predicted POS tags instead of gold POS tags. This undermines some of the results from those studies.

2.4 Linguistics and Neural NLP

In the previous section, I described a classical, pre-neural model of dependency parsing and then described how neural networks can be used to improve it. In the classical model, linguistic intuition is needed to construct useful features and their combinations. The role of linguistics in feature engineering for those types of models was emphasised by Hajičová (2011). The classifier, by contrast, is usually used as a black box which is expected to give appropriate weights to the features. In neural models, all parts can be seen as part of a black box: some models are trained end-to-end with very little information specific to the task. For example, Vinyals et al. (2015) treat parsing as a sequence-to-sequence task such as in machine translation where the source consists of the tokens of the sentence and the target is a linearised version of its parse tree. It is perhaps less clear where linguistic intuition is necessary or useful in this kind of model, besides providing annotated data. Hajičová (2011) said, when talking about statistical models:

> ‘The machine learner cannot be instructed “please use any context and any combination of features and tell me which are important” – there are simply too many of them.’

It seems that neural models are capable to learn feature combinations, as well as learn from a larger context window than before. We can ask ourselves if this means that linguistic knowledge has become irrelevant. In fact, one of the motivations for building neural models is to avoid relying too much on linguistically engineered features, as described in Collobert et al. (2011). There is some consensus among researchers that theoretical and computational linguistics have diverged, after a common history, with the rise of statistical methods, and could inform each other more (Pullum, 2009; Steedman, 2011). It may seem unintuitive at first to think that this situation will improve with the use of neural models, if we use them as black boxes. However, Manning (2015) believes that, on the contrary, having neural models for NLP will make it possible for the field to return to linguistic questions. He gives a few examples highlighting the potential for distributional representations of words to: 1) better explain human generalisation and 2) allow us to model large contexts. As we have seen in the previous section, RNNs indeed allow us to build models that do not rely on a strong Markov assumption.

Since the paper by Manning (2015), there have been many attempts at characterising what neural models learn about language, as well as building linguistic knowledge into neural models. In this section, I review some lines of work where linguistics can inform neural NLP. First, in Section 2.4.1, I discuss how linguistics can inform architecture engineering. Second, in Section 2.4.2, I discuss work that aims at analysing and interpreting what neural networks learn about language. Finally, I describe multi-task learning, in Section 2.4.3.

2.4.1 Architecture Engineering

Some work has been trying to incorporate linguistic knowledge as an inductive bias in neural model architectures. Kong (2017) shows how to add linguistically motivated inductive biases to models. He does that for three different models for three different tasks: segmenting, parsing and translation. For segmenting, he shows the importance of modelling segment structures rather than using a simple sequence labelling scheme. For parsing, he shows that a dynamically structured model which learns representations of subtrees is better than an attention mechanism. For translation, he explicitly models alignment decisions. In a similar vein, Yang (2019) studies the incorporation of structural biases in neural network models for different NLP tasks. For example, for document classification, he models the hierarchical structure of documents and incorporates two attention mechanisms, at the word and at the sentence level, to model the intuition that different words and sentences in a document are differently informative. This study is also published in Yang et al. (2016).

Battaglia et al. (2018) argue more generally (not just in the context of language processing) that deep learning models could benefit from better inductive biases, specifically making it possible to manipulate structured knowledge.

In the domain of syntax and parsing, Dyer (2017) stresses the importance of modelling hierarchical structure (similar to the parsing part of Kong (2017)).

The parser of Dyer et al. (2016) is an example where hierarchical structure is modelled: the representations of phrases are recursively composed through a composition function. This model will be described in Chapter 4. RNNs model **sequential** recency and Dyer (2017) argues that it is important to have a bias for **syntactic** recency. They find this composition function to be important, it has a large impact on accuracy on English and Chinese data. These results are further strengthened in Kuncoro et al. (2017) who show, using ablation experiments, that composition is key in the Recurrent Neural Network Grammar (RNNG) generative parser by Dyer et al. (2016).

2.4.2 Interpretability

A related line of work seeks to investigate the inductive biases of architectures by testing their capacity at learning a task. In Linzen et al. (2016), Enguehard et al. (2017) and Gulordava et al. (2018), language models are tested on agreement tasks to probe the syntactic abilities of LSTMs. The agreement task from Linzen et al. (2016) consists of predicting whether a verb should be plural or singular. They train an LSTM language model and test it on sentences with agreement. If the LSTM ranks the correct verb form higher than the incorrect form, the model prediction is considered correct for that example, otherwise it is considered incorrect. Linzen et al. (2016) argue that the model needs to be able to learn hierarchical information to be able to perform well on that task and, therefore, that its capacity at learning that task is a good indicator of the presence or absence of a structural bias in the model.

This line of work was extended in different ways: by looking at the capacities of different architectures and by looking at agreement in different languages or in synthetic data. Tran et al. (2018) compared LSTMs with Transformers (Vaswani et al., 2017) and Kuncoro et al. (2018) compared LSTMs with recursive neural networks. Gulordava et al. (2018) investigated agreement in four languages and Ravfogel et al. (2018) investigated the case of Basque. Gulordava et al. (2018) created what they call *nonce* sentences: sentences that are grammatically correct but nonsensical. This is to test if the models rely on lexical cues to learn the agreement task. Ravfogel et al. (2019) created synthetic data with different word orders, to test if word order has an impact for learning this task. The findings of these studies are relevant to Chapter 4 where they are discussed more at length.

In a similar fashion, Le Godais et al. (2017) test the lexical capacity of a character-level neural language model. In this work, the task is to decide whether or not a string of characters forms a word, and this can be evaluated for a language model by looking at whether it assigns more probability to a word than to a sequence of characters of the same length which is not a word. Other examples of tasks used to test capacities of LSTMs are filler-gap dependencies (Wilcox et al., 2018) and grammaticality (Marvin and Linzen, 2018).

The studies presented so far in this section investigate the capacities of neural networks. A related effort seeks to interpret more precisely what kind of information the networks learn and where they learn it, for example in which layer of the network. These studies use prediction tasks to probe what information is encoded in specific vectors (for example, sentence vectors in Adi et al. (2017)). The vectors under investigation are used to train a classifier. If a classifier can be trained on a set of vectors for a task, this is an indication that the vector contains information that is relevant for the task. Adi et al. (2017) investigate whether sentence embeddings obtained with different techniques encode information about sentence length, about the words that constitute the sentence and about word order in the sentence. A similar method has been used to investigate machine translation models (Belinkov et al., 2017; Dalvi et al., 2017; Shi et al., 2016) and speech models (Belinkov and Glass, 2017). This method is called *auxiliary prediction tasks* by Adi et al. (2017) but has received other names. It has been called *probing tasks* by Ettinger et al. (2016) and *diagnostic classifiers* by, among others, Hupkes et al. (2018), which is the term I adopt in this thesis.

Belinkov and Glass (2019) provide a more extensive list of papers using the methods presented in this section and a more detailed survey of general methods for analysing models in neural NLP.

2.4.3 Multi-task Learning

A trend which is increasing in neural NLP (and machine learning in general) is to train tasks jointly, which has been called multi-task learning (MTL). In multi-task learning, a network is trained with several losses corresponding to several tasks and at least some of the parameters of the models are shared. In Figure 2.15, the lower layers of the network are shared and there is a task specific layer for each task. If we take the parsing model described in Section 2.3.2 as an example of Task A, the lower layers would correspond to the word and POS embeddings and the BiLSTM and the task-specific layer for parsing would be the MLP. That model could be trained jointly with for example POS tagging as Task B where we would need a separate layer to predict the POS tag, for example a softmax layer.

Many studies have shown the benefit of MTL for various NLP tasks, starting with Collobert and Weston (2008) who improved semantic role labelling by training it jointly with several other tasks, separately or combined: POS tagging, chunking, named entity recognition, semantic similarity and grammaticality judgment. They obtained significant improvements on semantic role labelling with all variations of their joint models. There are many different ways to share the parameters of the network. I describe different strategies in Chapter 6. For a more complete overview of MTL, see Ruder (2016).

Linguistic intuition can be useful in designing the MTL network architec-

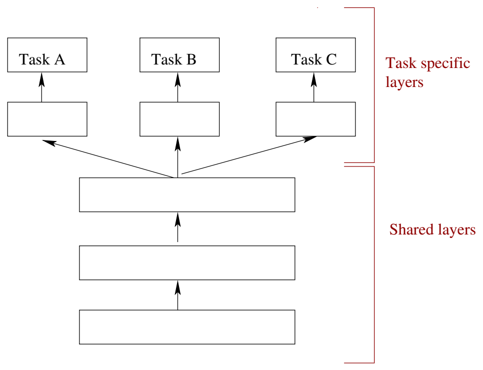


Figure 2.15. MTL architecture with three tasks.

ture. For example, Søgaard and Goldberg (2016) train a network for syntactic chunking, CCG supertagging and POS tagging and find that having the POS tagging supervision layer at a low layer is most beneficial, which makes sense linguistically since POS tagging is a lower-level task than the other two. They train a model similar to the one we presented for parsing: with a word embedding layer, with BiLSTMs and with an output layer (in this case, three output layers). They use three layers of BiLSTMs and show that using the POS tagging output layer at the output of the first BiLSTM works better than having it at the output of the third BiLSTM.

In Section 2.4.1, we proposed that linguistic intuition can be used to guide architecture engineering. In this section, we propose that linguistic intuition can be used to guide MTL architecture. This can also be seen as a special case of architecture engineering.

I introduced this section by asking the question of whether or not linguistic intuition can be used in neural NLP. Linguistic intuition was useful when manually designing features in pre-neural NLP, but this part has been automated. I have described three ways in which linguistic intuition can be used in neural NLP. We can use linguistic knowledge to guide architecture engineering. We can use it to interpret what neural models learn about language. Finally, we can use it to design multi-task learning architectures where different tasks inform each other. This can be seen as a special case of architecture engineering. We can also combine these approaches. For example, based on the results from Linzen et al. (2016) where LSTM language models were shown not to learn agreement without explicit supervision, Enguehard et al. (2017) propose to use MTL where agreement is used as an auxiliary task to language modelling and find out that this does make it possible to learn the task. This list is not exhaustive but it seems clear that there is a lot that we can do to improve neural models with the use of linguistic knowledge.

2.5 Conclusion

This Chapter started with a description of UD and the research opportunities that it opens up: 1) the possibility to incorporate linguistic knowledge into our models while making our models generalisable across languages more than was possible before and 2) leveraging resources from different languages to improve parsing on the low-resource ones. I described an essential property of UD which is the primacy of content words. I described how this property means that the representation of UD trees is compatible with their representation in Tesnière (1959), and the representations in Tesnière (1959) are richer than the representations which have been used so far in dependency parsing. The UD representation is compatible with an interpretation where the relationship between the head and the dependent of, for example, an auxiliary dependency relation, is a relation of transfer, and the two nodes of that relation form a dissociated nucleus. This is linguistic knowledge that has not been incorporated into our parsing models and I consider this in Chapter 5.

I went on to describe dependency parsing and its recent developments. I focused on transition-based parsing which can be decomposed into two main components: the feature extraction step which extracts features representing the current configuration, and the classifier which scores the possible transitions. I described how these components work in pre-neural parsing and how they can be replaced by neural networks to improve parsing accuracy. The linear classifier is replaced by a non-linear multilayer perceptron. The feature extraction function can be improved by the use of LSTMs which capture information about words in an unbounded window. Neural parsing is also improved by replacing high-dimensional feature vectors by dense vectors representing core features. I described the popular architecture by Kiperwasser and Goldberg (2016b) and its extensions for the graph-based parser by Dozat and Manning (2017) and then Dozat et al. (2017). We extend the transition-based parser of Kiperwasser and Goldberg (2016b) in similar ways to what was done in Dozat et al. (2017) to make it better for UD treebanks: by adding character representations but also by allowing the construction of non-projective trees. These extensions are described in Chapter 3. With these investigations, we attempt to provide answers to **RQ1**: *How can we build parsing models that perform well across typologically diverse languages?*

I finally described ways in which linguistic information can be incorporated into neural models, with the aim to provide answers the overarching question of this thesis: **RQ0**: *How can linguistics inform neural NLP?* I identified three methods. The first is architecture engineering where linguistic knowledge is used to guide the neural network architecture. I discussed work where the importance of incorporating hierarchy in neural NLP models was highlighted. In Chapter 4, we re-evaluate that line of work by looking at a simpler architecture, the architecture by Kiperwasser and Goldberg (2016b) and by investigating this phenomenon with more languages than done previously. This is an attempt at investigating **RQ2**: *How can we incorporate linguistic knowledge into neural models for dependency parsing?*

The second method to use linguistics to inform neural NLP is to design tasks which probe what they learn. I attempt to find out whether a neural dependency model learns the notion of dissociated nucleus in Chapter 5. This is an attempt to provide an answer to **RQ3**: *How can we investigate what neural models learn about language when trained for the task of dependency parsing?* I also investigate whether or not a hierarchical bias in the model can help learn this notion, thereby further exploring the results of Chapter 4 and **RQ2**.

The last method is to make use of multi-task learning where linguistic knowledge can be used to guide the design of the MTL architecture. Polyglot NLP can be treated as a multi-task learning problem such as is done in Guo et al. (2016) who exploit MTL for dependency parsing across languages.[10] They create one network to parse several languages with some of the parameters that are shared, the ones that they think should be language-independent, and some are language-specific. We explore this idea further in Chapter 6, in an attempt to provide answers to **RQ4**: *How can we leverage information from multiple treebanks in different languages to improve parsing performance on the individual languages?*

While we use a polyglot setting to answer the last research question, we use a polymonolingual setting for the others. This allows us to study the effect of parser design on typologically diverse languages. Having some understanding of how parsing design principles interact with different typological properties should give insights that are useful when working with polyglot parsing.

[10]Note that Ruder (2019) considers cross-lingual learning and multi-task learning as two different types of transfer learning.

3. Parsing Typologically Diverse Languages

This chapter describes concerns with building models that are suitable for typologically diverse languages. With this, I aim to answer **RQ1**: *How can we build parsing models that perform well across typologically diverse languages?*

A first concern when working with UD is that the distribution of resources is not representative of the distribution of the world's languages. There is therefore a concern that our models will be biased towards the language families that are overrepresented. An additional difficulty when working with UD and neural network models is that it is expensive to train a model for each treebank. Our solution to these two issues which I present in Section 3.1 is to select a sample of treebanks which is hopefully representative of the world languages while having a practically useful size.

Bender (2011) gives recommendations for making NLP models typologically aware. She argues that if a system which claims to be language independent is better for one language compared to another, it means that it is making assumptions that fit more the language for which it is better than the other. She gives the example of n-gram models which work best for languages that do not have complex morphology. Parsing scores also tend to be lower for more morphologically complex languages than for others. This can be explained by lexical data sparsity: morphologically complex languages have more morphological variations of surface forms than languages with simpler morphology. As described in Chapter 2, UD assumes that words are the basic units of syntax which means that the statistics of word forms are crucial for parsing UD treebanks. Recent work has shown the benefit of using character models to mitigate the problem of lexical data sparsity. Character models were shown to improve parsing accuracy more for languages with more complex morphology (Ballesteros et al., 2015). We further investigate the use of these character models in our parsing model in Section 3.2.

Models developed for English work well when assuming that trees should be projective. These models perform much worse on languages whose treebanks have a high ratio of non-projective arcs. I describe an existing method for dealing with non-projectivity in transition-based parsing and our extension to this solution to keep the benefits of dynamic oracles.

Our methods are evaluated individually in the respective sections. I give a more comprehensive evaluation of our parser in Section 3.4 which presents our results from the CoNLL 2017 and 2018 shared tasks. The concerns addressed in this chapter are directly relevant for the development of polymonolingual models (Chapter 4 and 5) but also for polyglot parsing (Chapter 6).

3.1 Treebank Sampling

As argued in the previous chapters of this thesis, UD is a resource that makes it possible to build parsing models that will generalise better than if we build our parsing models with a restricted set of languages and datasets. However, a concern when working with UD is that a large part of the treebanks represents a small portion of the world's languages. In particular, Indo-European languages are highly represented compared to other language families: they represent 45 of the 83 languages in v2.4 and contain the largest treebanks. There is therefore a concern if we work with these treebanks that our models will be biased towards these overrepresented languages. The question I ask in this section is therefore the following:

RQ1a How can we develop and evaluate models with a large dataset such as UD?

In the context of research in typology, Velupillai (2012) points out that we can never work on all languages of the world because the majority of world languages are not documented (and even if they were, it would be technically complicated to study all languages together). She therefore suggests that we inevitably have to work with language samples and that we should make sure to work with a sample that is as unbiased as possible. The set of UD treebanks is clearly a biased sample and it therefore seems beneficial to downsample it to mitigate this bias problem.

Additionally, a disadvantage of working with a large set of treebanks such as UD is that it is expensive to train models for the full set of treebanks, especially with neural network models which require learning a substantial amount of parameters. As a matter of fact, even though as explained in the background section, the parser by Chen and Manning (2014) is faster than its classical counterpart at prediction time, it is slower at training time. Gylling (2017) reports training times for a classical parser compared to a parser using a feed-forward network as a classifier and finds that the latter takes three times as much time as the former. The parsers that use BiLSTMs have substantially more parameters than the ones that do not and are slower both at training and prediction time. As UD grows, it may become increasingly prohibitive to train models for the full set of treebanks when doing parser development. This is another reason to downsample the set of UD treebanks.

We suggest to sample a subset of UD treebanks to do parser development where we can test small variations in the parsing architecture or features. This way we can inspect the behaviour of the parser in detail and formulate reasonable hypotheses on what methods work best. Ideally, we should keep held-out languages for testing these hypotheses in a subsequent step. The idea to keep held-out languages for evaluation was proposed by Bender (2011). For this however, we would need to be able to construct two representative samples, one for development and one for evaluation. This is not necessarily an easy thing to do with the current state of UD treebanks, as will become clear in the discussion that follows. However, as UD grows and as annotation quality improves, this might become easier. For now, we stick to one sample in our studies. Evaluating on the full set of treebanks can also be a useful thing to do since having results for a large number of languages makes it easier to study correlations between language properties and parsing accuracy than when using a sample. We do this in the context of the CoNLL 2017 and 2018 shared tasks. The shared tasks additionally allow us to compare our system to a large number of different systems.

We propose criteria to select a sample of treebanks. The objective is to have a sample as representative of world languages as possible. Bender (2011) emphasises the importance of working with data from different language families to make sure that our models generalise. Velupillai (2012) also emphasises the importance of avoiding to have a sample with what she calls a *genetic bias* where some language families are overrepresented. We therefore first want to ensure typological variety. UD languages can be divided into language families and genera, as defined in WALS (Dryer and Haspelmath, 2013). For example, the language family for Swedish is Indo-European and its genus is Germanic. Information about those families can be found in WALS online (Dryer and Haspelmath, 2013). We make it a requirement to not select two languages from the same genus and we also make sure to have some variety in language families. Bender (2011) argues that it is beneficial to ensure diversity in morphological complexity. For this reason, we set a criterion to have diversity in morphological complexity (we seek to include at least an isolating language, an agglutinative language, and a language in between). We additionally need to ensure variability of treebank sizes and domains, since we know that machine learning models are sensitive to data size[1] and we know that differences in domains can have a high impact on parsing accuracy.[2] Since parsing non-projective trees is notoriously harder than parsing projective trees, if we want to evaluate a system, we also need to make sure to have at least one treebank with a large amount of non-projective trees. It is finally useful to take the quality of treebanks into account, in particular, there are known issues[3] about inconsistency in the annotation. It is best to select languages that have as little of those as possible to avoid relying on noisy data.

[1]For example Koo et al. (2008) showed large drops in accuracy when training a parser on a small portion of a treebank.

[2]McClosky et al. (2010), among others, have shown a large drop in accuracy when training and testing a parser on different domains.

[3]At the time of doing that work, issues were reported online at `http://universaldependencies.org/svalidation.html`. This is currently not the case, but annotation can be validated with the official script, available at `https://github.com/UniversalDependencies/tools/blob/master/validate.py`.

The criteria can be summarised as such:

1. Typological variety
   a) Only different genera and as many families as possible
   b) Diversity in morphological complexity
   c) One treebank with high non-projective arcs ratio
2. Variety in treebank sizes and domains
3. High annotation quality

This set of criteria is defined generally and can result in different samples depending on how they are applied. The application of these criteria is not straightforward due to the properties of UD treebanks. The majority of treebanks are Indo-European and treebanks from Indo-European languages include the largest part of high-resource languages. This makes it difficult to select languages from different families while having treebanks with a decent size. In addition, it is difficult to include a variety of domains, a large part of UD treebanks contain only newspaper text. Finally, most of the treebanks have not been annotated natively with the UD guidelines but have been converted from a treebank annotated with different guidelines. This is the case for example of the Czech PDT treebank, a conversion of the Prague Dependency Treebank (Hajic et al., 2001). The criteria therefore have to be balanced against each other and different studies might prioritise differently. For this reason, constructing two samples, one for development and one for evaluation, is difficult at this point. However, doing this may become feasible as UD grows and as annotation quality improves.

We do the selection manually. This is because we do not have direct access to some of the information: treebanks are not annotated for morphological complexity, and the annotation quality cannot easily be reduced to one score. There are different ways in which we can automatically measure morphological complexity, the simplest being the type-token ratio. Some of these measures have been evaluated on UD treebanks by, among others, Berdicevskis et al. (2018) who evaluated their robustness. If we had a way to measure annotation quality, we could automate the sample selection process. This would require defining the criteria and their interaction precisely, which is also not trivial. We could also add an element of randomness to make sure we do not always work with the same languages, which would be beneficial to avoid overfitting.

An example of how these criteria can be applied can be found in de Lhoneux et al. (2017c) where we applied them to UD version 1.3 (Nivre et al., 2016b) using a sample size of 8. In that study, our objective was to do an extensive comparison of two parsers, MaltParser (Nivre et al., 2006) and UDPipe (Straka et al., 2016). These parsers use morphological features which were not available for all treebanks in that version of UD, and we therefore had to add a criterion, the availability of morphological features, to ensure comparability. UD 1.3 contains a total of 15 different genera and 8 language families. We selected 8 treebanks with 6 different families. The selection is given in Table 3.1 together with main arguments for inclusion for each.

Table 3.1. *Treebank Sample. IE means Indo-European. In parenthesis, the criterion that applies. TB means treebank.*

language	family	genus	main argument for inclusion
Czech	IE	Slavic	largest TB (2)
Chinese	Sino-tibetan	Sinitic	the only isolating language (without copyrights) (1b)
Finnish	Uralic	Finnic	agglutinative (1b); has many different domains (2)
English	IE	Germanic	largest TB with full manual check of the data (3)
Ancient Greek	IE	Hellenic	large percentage of non-projective trees (1c)
Kazakh	Turkic	N.western	smallest TB (2); full annotation manual check (3)
Tamil	Dravidian	Southern	small TB (2); language family (1a)
Hebrew	Afro-Asiatic	Semitic	complex morphology/language family (1a and 1b)

The solution of sampling is our answer to **RQ1a**: *How can we develop and evaluate models with a large dataset such as UD?* Our sampling criteria are designed with the objective to develop and evaluate parsing systems. They can be used for different objectives in which case they may need to be adapted. The criterion to have one treebank with a large ratio of non-projectivity is only relevant when comparing parsing accuracy of two systems, and is not necessarily relevant when analysing what models learn such as is done in Chapter 5. The criterion to include different data sizes is much more relevant for parser evaluation than for parser development and analysis. We use the sampling technique in different studies in this thesis and I discuss more how these criteria are adapted when describing those studies.

The method of sampling allows us to be aware of typological diversity when designing models and test the impact of design assumptions on different languages. This is not a perfect way to do this because there are various factors at play. Gulordava and Merlo (2016) propose a method to decouple properties of treebanks from properties of languages by creating synthetic treebanks and evaluate parsers on data with systematic permutations of linguistic properties. This is an interesting avenue for research which we do not explore here.

3.2 Character Models

Ballesteros et al. (2015) have shown with their stack LSTM parser (Dyer et al., 2015) that an LSTM can be used to construct a vector representing the characters of the word and that this character vector is useful for parsing. They replace the ordinary representation of word forms in their model by this character vector and show that doing so improves parsing accuracy, especially for morphologically complex languages. They also find that character vectors make the use of POS information less necessary: adding a POS embedding improves a model with word type vectors much more than a model with character vectors. We conclude from their results that character information is more robustly useful across languages than POS tag information. It is still an open question whether that means that character models trained as part of a parser learn mor-

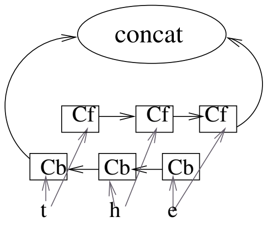


Figure 3.1. Character BiLSTM over the word *the*.

phology. Vania et al. (2018) investigate this question and find that these models do learn some amount of morphology but parsers can still benefit from explicit morphological supervision, especially for case marking.

Inspired by this success, we integrate character information into our parsing architecture and verify that the findings of Ballesteros et al. (2015) holds with our architecture. We seek to answer the following question:

RQ1b Is character information robustly useful across languages in a BiLSTM-based parsing architecture?

We modify the word representation at the input of the BiLSTM in a K&G parser, which was described in Section 2.3.2 and is the base parser used in this thesis. As illustrated in Figure 3.1, we construct a character vector $ce(w_i)$ for each w_i by running a BiLSTM over the characters ch_j $(1 \leq j \leq m)$ of w_i:

$$ce(w_i) = \text{BiLSTM}(ch_{1:m}) \tag{3.1}$$

We concatenate that character vector to the representation of the word at the input of the (word) BiLSTM. We test 4 variations of the parser. In the full model, the network input at position i, x_i, is a concatenation of a word embedding $e(w_i)$,[4] a character embedding $ce(w_i)$ and an embedding of the word's POS tag $e(t_i)$. The input vectors x_i of the full model are therefore:

$$x_i = [e(w_i); ce(w_i); e(t_i)] \tag{3.2}$$

In the other models, we remove either or both of the character and POS vectors. We apply the criteria from Section 3.1 to sample treebanks from a more recent UD version than in that section: v2.1 (Nivre et al., 2017b). This gives us the possibility to construct a more diverse selection than in that section. We do not include small treebanks which are more relevant for parser evaluation than development as is the case here. Our sample is: Ancient Greek (PROIEL), Basque, Chinese, Czech, English, Finnish, French, Hebrew and Japanese. Results are given in Figure 3.2. As is consistent with the results from Ballesteros

[4]This word embedding is randomly initialised. When we initialise word embeddings with pre-trained embeddings, I specify it, and these embeddings are denoted as $pe(w_i)$.

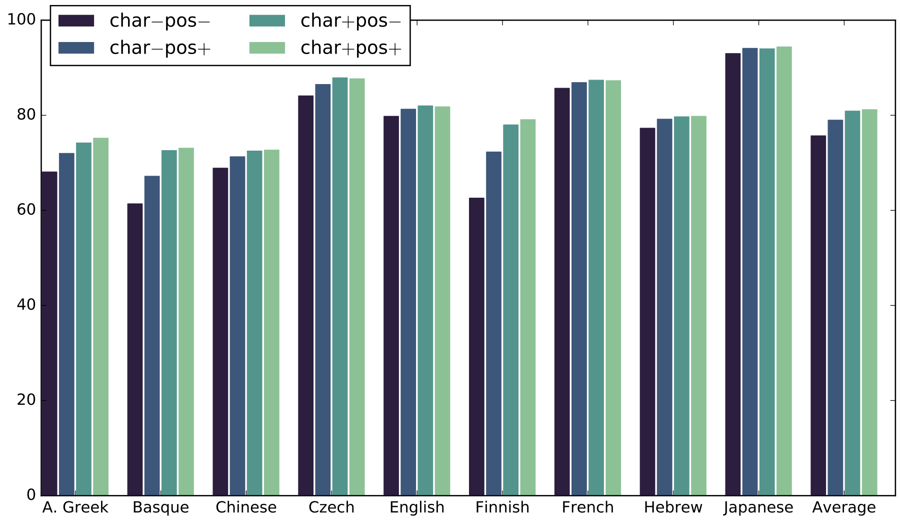


Figure 3.2. LAS of baseline, using character and/or POS tags to construct word representations.

et al. (2015) and Smith et al. (2018b), using character-based word representations and/or POS tags consistently improves parsing accuracy but has a different impact with different languages and the benefits of both methods are not cumulative. We see bigger gains from using character representations for morphologically complex languages than for others: Finnish improves the most, followed by Basque. POS tags consistently improve parsing accuracy but consistently less than character representations do. In addition, in most cases, the model using characters performs almost as well as the model using both characters and POS tags and in 2 cases (Czech and English), it actually performs better. We have therefore verified that character models are an efficient way to obtain large improvements for morphologically complex languages, and it is overall a more robustly useful method of constructing vectors representing words than POS tags. This answers **RQ1b** positively: *Is character information robustly useful across languages in a BiLSTM-based parsing architecture?* To keep the model simple and to minimise the number of confounding variables, we mostly use a parser where the representation of words is simply a representation of the word form and its characters ($x_i = [e(w_i); ce(w_i)]$) in the remainder of this thesis. We use POS tags only where relevant and indicate when we do. Our model with this addition as well as further modifications is more extensively evaluated in Section 3.4.

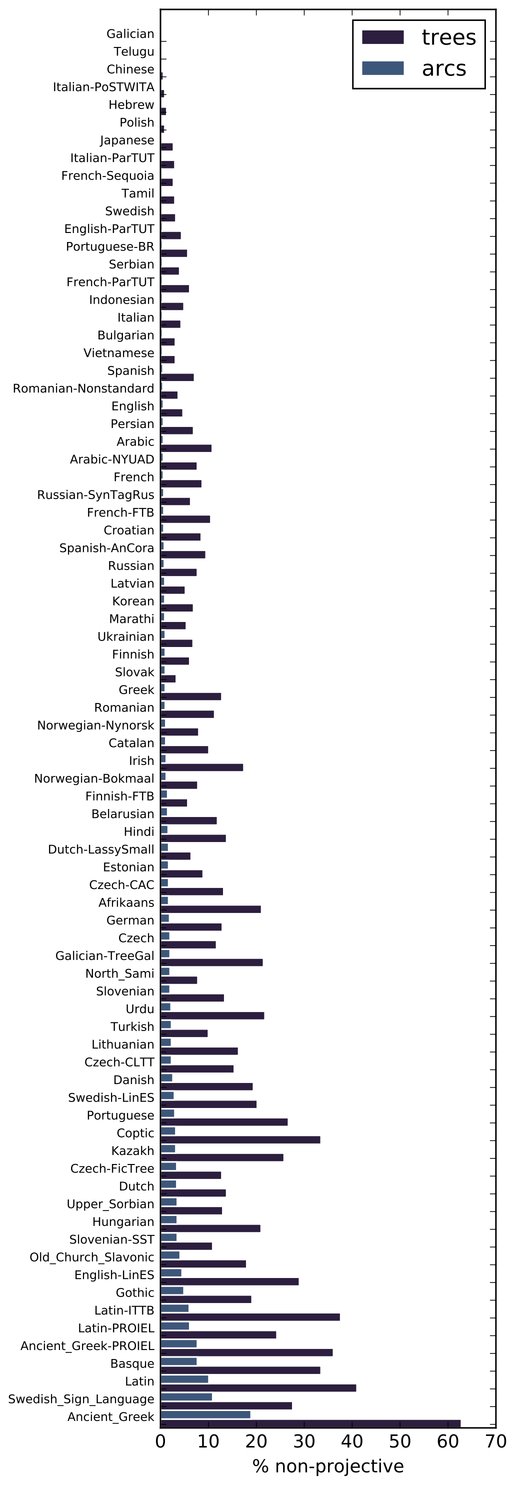


Figure 3.3. Non-projective arc and tree percentage in UD 2.1 treebanks.

3.3 Non-Projectivity

As described in Chapter 2, a non-projective tree is a tree with crossing arcs. Non-projective trees are harder to deal with than projective trees. For this reason, early work in transition-based dependency parsing (Nivre, 2003, 2004) restricted the construction of dependency trees to projective ones. Figure 3.3 illustrates non-projectivity statistics collected from the training sets in UD version 2.1, sorted by increasing percentage of non-projective arcs. The detailed statistics are reported in Appendix B. We can see that the ratio of non-projective trees and arcs varies a lot in the different treebanks, ranging from no non-projectivity at all (Galician) to 18.9 percent non-projective arcs and a majority of non-projective trees in Ancient Greek. This shows that there is little cost (it does not hurt parsing accuracy) to restricting parsing to projective trees when working with the languages that are near the top of the figure, including English[5] and Chinese, but the cost can be very high when working with a language like Ancient Greek. The transition-based parser by Kiperwasser and Goldberg (2016b) can only produce projective arcs and therefore is expected to perform poorly on highly non-projective treebanks. There are different methods for dealing with non-projectivity in transition-based parsing with different advantages and disadvantages. The question we seek to answer here is the following:

RQ1c How can we deal with non-projectivity in transition-based parsing in a convenient way?

I describe ways to deal with non-projectivity in transition-based parsing in Section 3.3.1. This includes pseudo-projective parsing, which we used for the CoNLL shared task 2017 whose result I present in Section 3.4, as well as the solution to use reordering which is used in the remainder of this thesis. As mentioned in Section 2.3.1, the training procedure that makes use of a static oracle is suboptimal and the use of dynamic oracles has been shown to improve transition-based parsing. I describe dynamic oracles in Section 3.3.2. Combining the reordering approach with dynamic oracles is a non-trivial task for which we propose a partial solution in Section 3.3.3.

3.3.1 Non-projective Transition-Based Parsing

There are various ways to allow the construction of non-projective trees in dependency parsing. One possibility is to do pseudo-projective parsing. Pseudo-projective parsing, as described by Nivre and Nilsson (2005), consists of a pre-processing and a post-processing step. The pre-processing step consists in projectivising the training data by reattaching some of the dependents higher in the tree, to a close parent, and the post-processing step attempts to deprojectivise trees in output parsed data. In order for information about non-projectivity

[5]The LinES treebank seems to be an exception here.

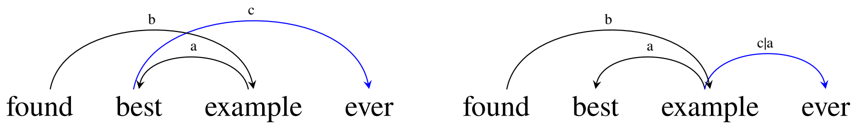


Figure 3.4. Pseudo-projectivisation of a tree. Arc modified in blue.

Initialisation:	$c_0(x = (w_1, \ldots, w_n)) = ([\,], [1, \ldots, n, 0], \emptyset)$	
Termination:	$C_t = \{c \in C \mid c = ([\,], [0], A)\}$	
Transition		**Condition**
LEFT-ARC$_d$	$(\sigma\|i, j\|\beta, A) \Rightarrow (\sigma, j\|\beta, A \cup \{(j, r, i)\})$	$j \neq 0 \vee \sigma = [\,]$
RIGHT-ARC$_d$	$(\sigma\|i\|j, \beta, A) \Rightarrow (\sigma\|i, \beta, A \cup \{(i, r, j)\})$	
SHIFT	$(\sigma, i\|\beta, A) \Rightarrow (\sigma\|i, \beta, A)$	$i \neq 0$
SWAP	$(\sigma\|s_0,\ b\|\beta,\ A) \Rightarrow (\sigma,\ b\|s_0\|\beta,\ A)$	$\beta \neq 0 \wedge s_0 < b$

Figure 3.5. Transitions for the arc-hybrid transition system with an artificial root node (0) at the end of the sentence. The stack Σ is represented as a list with its head to the right (and tail σ) and the buffer B as a list with its head to the left (and tail β).

to be recoverable after parsing, when projectivising, arcs are renamed to encode information about the original parent of dependents which get re-attached. There are a few different ways of doing that, including what Nivre and Nilsson (2005) call the *head* schema. Figure 3.4 gives an example of using this *head* schema. The left part of the figure is our non-projective tree from before and the right part is its pseudo-projective version. As mentioned previously, the non-projective arc is the arc between the dependent word *ever* and its head *best*. The dependent *ever* gets re-attached to the closest ancestor which makes the tree projective. In this case, it is only moved one level up the tree and attached to the head of its head, *example*. The label gets updated with the label of its original head *best*: *a*, which is concatenated to the original label of *ever*, *c*, to form $c|a$. We adopted this method in the CoNLL 2017 shared task, as described in Section 3.4

Although this technique seems to be an adequate way of dealing with non-projectivity, it has some disadvantages. A practical disadvantage is that it requires a pre- and a post-processing system. Such a system can also lead to error propagation. Another disadvantage is that this method increases the size of the output label set, which means that the data gets sparser and it might be a problem with small treebanks. As a matter of fact, Nivre and Nilsson (2005) report in a footnote that this was a problem with the Danish Dependency Treebank which has 100K words, a lot more than many UD treebanks. This means that this method is not adequate for those treebanks.

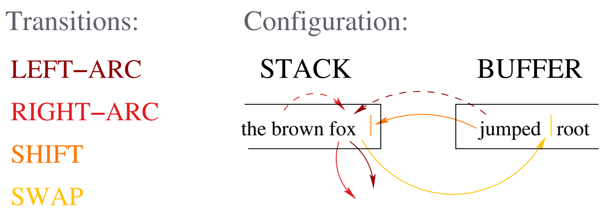


Figure 3.6. Arc-hybrid transitions with SWAP. A dashed arrow represents an arc and a solid arrow represents movement of an element from the stack or buffer.

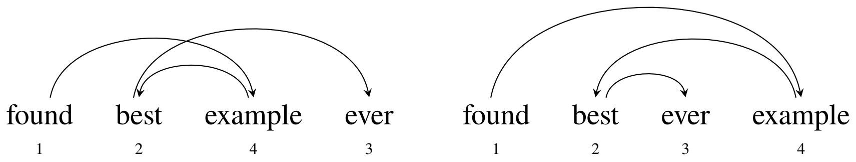


Figure 3.7. Non-projective tree with canonical projective order (left tree) and its reordered version (right tree).

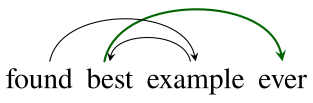
Another method by Nivre (2009) which we therefore explore in this section and which is used in the remainder of this thesis is to make use of reordering. Nivre (2009) defines a canonical way of reordering nodes in a tree to make it projective: through an inorder traversal of the tree that respects the local order of a head and its dependents. This order is called the *projective order*. He extends a transition system, in this case the arc-standard system, with an additional transition SWAP which allows reordering the nodes by moving a node from the stack to the buffer. We define this extension for the arc-hybrid system which has a desirable property which arc-standard lacks, as explained in Section 3.3.2. We define the projective version of the arc-hybrid system in Figure 2.9 and extend it in Figure 3.5.[6] We add a SWAP transition which moves the top item of the stack s_0 to the second position in the buffer, thus inverting the order of s_0 and b. There is a precondition making SWAP legal only if the stack is not empty, the buffer contains at least two elements and s_0 and b are not in projective order, i.e. the projective order of b is higher than the projective order of s_0.[7] The transitions are illustrated in Figure 3.6. The canonical projective order of our non-projective tree from Figure 2.8 is given in Figure 3.7. If the nodes are reordered as in the right part of the figure, the tree is projective. The transition sequence for that tree is given in Figure 3.8. In that sentence, only one reordering is needed: *ever* and *example* need to be swapped. First, *example* is shifted, in transition 3. Then it is swapped in transition 4, at which

[6]This extension includes a modification we made to disallow constructing trees with multiple roots. This is explained in Section 3.4.

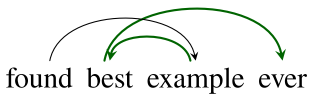
[7]The last condition is needed to guarantee termination: it prevents two words from being swapped more than once.

point the tree is in projective order and parsing can continue as normal. Nivre (2009) proved that this method allows us to construct any non-projective tree when working with arc-standard.

0		[ ]$_\Sigma$	[found$_1$ best$_2$ example$_4$ ever$_3$]$_B$
1	SHIFT	[found$_1$]$_\Sigma$	[best$_2$ example$_4$ ever$_3$]$_B$
2	SHIFT	[found$_1$ best$_2$]$_\Sigma$	[example$_4$ ever$_3$]$_B$
3	SHIFT	[found$_1$ best$_2$ **example**$_4$]$_\Sigma$	[ever$_3$]$_B$
4	SWAP	[found$_1$ best$_2$]$_\Sigma$	[ever$_3$ **example**$_4$]$_B$
5	SHIFT	[found$_1$ best$_2$ ever$_3$]$_\Sigma$	[example$_4$]$_B$
6	RIGHT-ARC	[found$_1$ best$_2$]$_\Sigma$	[example$_4$]$_B$



7	LEFT-ARC	[found$_1$]$_\Sigma$	[example$_4$]$_B$



8	SHIFT	[found$_1$ example$_4$]$_\Sigma$	[ ]$_B$
9	RIGHT-ARC	[found$_1$]$_\Sigma$	[ ]$_B$

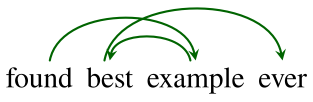


Figure 3.8. Transition sequence for a non-projective tree. Subscript numbers are the projective order of the word. A thick green arc is an arc that has been constructed.

This transition systems increases the worst case complexity of parsing from $O(n)$ (since 2 actions are needed for every word: a SHIFT and either a LEFT-ARC or a RIGHT-ARC transition) to $O(n^2)$, since words can be swapped n times at most (the condition that s_0 and b are not in projective order prevents a pair of words from being swapped twice). However, Nivre (2009) has shown empirically that we can expect linear parsing time, because SWAP operations are rare. He has shown this to be true for the arc-standard system and there is no reason to believe that this conclusion will not hold for arc-hybrid. Other methods for non-projective transition-based parsing not discussed here can be found in Attardi (2006), Nivre (2007) and Gómez-Rodríguez and Nivre (2010).

3.3.2 Dynamic Oracles

In Chapter 2, I presented a training procedure using a static oracle, repeated in Algorithm 2, where the static oracle is a function $o(c, T)$ which returns the

Algorithm 2 Transition-based parser training algorithm with a static oracle

```
1: INIT(w)
2: for $i \leftarrow$ ITERATIONS do
3:     for sentence $s$ with gold tree $T$ in corpus do
4:         $c \leftarrow c_s(s)$
5:         while $c$ is not terminal do
6:             $t_p \leftarrow \arg\max_t \mathbf{w} \cdot \phi(c,t)$
7:             $t_o \leftarrow o(c,T)$
8:             if $t_p \neq t_o$ then
9:                 UPDATE(w, $\phi(c,t_o), \phi(c,t_p)$)
10:            $c \leftarrow t_o(c)$
11: return w
```

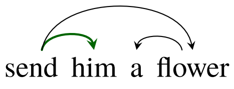
correct transition given a configuration c and the gold tree T. In training, we always follow the correct transition t_o given by the oracle. I mentioned the limitations of such a training procedure. First, there can sometimes be several correct transitions for a configuration but the oracle deterministically returns one. For example, I give two possible transition sequences with arc-hybrid and our projective sentence from Figure 2.8 in Figure 3.9. With an eager oracle as in the top part of the figure, we attach nodes as soon as possible, which means that the word *him* is attached as soon as it is allowed, in transition 3. It is however also possible to postpone this attachment as in the bottom part of the figure where it happens in transition 5, after shifting the word *a* and attaching it to the word *flower*.

This oracle considers the transition sequence in the bottom part of the figure as incorrect, since it only considers one transition sequence as the gold sequence, the one defined by the eager oracle in this case.

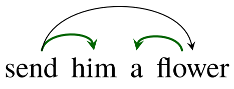
The second problem with training with a static oracle is that the parser only encounters configurations that result from taking correct transitions which means that it does not know how to recover from errors. At prediction time, this method therefore suffers from error propagation. As mentioned in Chapter 2, an improvement over this training procedure is to make use of dynamic oracles instead of this static oracle, as proposed by Goldberg and Nivre (2012) and Goldberg and Nivre (2013). Dynamic oracles define optimal transitions given any configuration. This allows exploring errors during training. More specifically, the training procedure is changed as in Algorithm 3. First, in line 8, we consider all correct transitions. Second, in line 11, we allow exploring errors: we follow a transition which we know is not optimal with some probability p.

The set of correct transitions CORRECT(c) considered in line 8 is defined by a boolean function $o(t;c,T)$ over the set of transitions for a configuration c which returns true for every transition t where the transition is optimal given the gold tree T, as defined in line 6. To define whether a transition is optimal given any

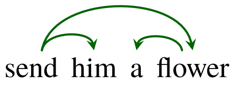
0		[ ]$_\Sigma$	[send him a flower]$_B$
1	SHIFT	[send]$_\Sigma$	[him a flower]$_B$
2	SHIFT	[send him]$_\Sigma$	[a flower]$_B$
3	**RIGHT-ARC**	[**send**]$_\Sigma$	[**a flower**]$_B$



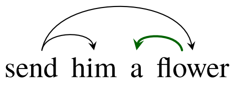
4	**SHIFT**	[**send a**]$_\Sigma$	[**flower**]$_B$
5	**LEFT-ARC**	[**send**]$_\Sigma$	[**flower**]$_B$



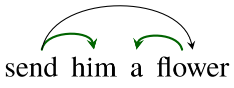
6	SHIFT	[send flower]$_\Sigma$	[ ]$_B$
7	RIGHT-ARC	[send]$_\Sigma$	[ ]$_B$



0		[ ]$_\Sigma$	[send him a flower]$_B$
1	SHIFT	[send]$_\Sigma$	[him a flower]$_B$
2	SHIFT	[send him]$_\Sigma$	[a flower]$_B$
3	**SHIFT**	[**send him a**]$_\Sigma$	[**flower**]$_B$
4	**LEFT-ARC**	[**send him**]$_\Sigma$	[**flower**]$_B$



5	**RIGHT-ARC**	[**send**]$_\Sigma$	[**flower**]$_B$



6	SHIFT	[send flower]$_\Sigma$	[ ]$_B$
7	RIGHT-ARC	[send]$_\Sigma$	[ ]$_B$

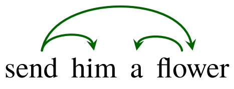
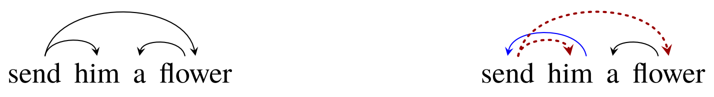


Figure 3.9. Two transition sequences for a projective tree. Different transitions in bold. A thick green arc is an arc that has been constructed.

Algorithm 3 Transition-based parser training algorithm with a dynamic oracle

```
1: $\textsc{Init}(\mathbf{w})$
2: for $i \leftarrow$ ITERATIONS do
3:     for sentence $s$ with gold tree $T$ in corpus do
4:         $c \leftarrow c_s(s)$
5:         while $c$ is not terminal do
6:             $\textsc{Correct}(c) \leftarrow \{t \mid o(t;c,T) = \textbf{true}\}$
7:             $t_p \leftarrow \arg\max_{t \in \textsc{Legal}(c)} \mathbf{w} \cdot \phi(c,t)$
8:             $t_o \leftarrow \arg\max_{t \in \textsc{Correct}(c)} \mathbf{w} \cdot \phi(c,t)$
9:             if $t_p \notin \textsc{Correct}(c)$ then
10:                $\textsc{Update}(\mathbf{w}, \phi(c,t_o), \phi(c,t_p))$
11:                $c \leftarrow \textsc{Explore}_{k,p}(c,t_o,t_p,i)$
12:            else
13:                $c \leftarrow t_p(c)$
14: function $\textsc{Explore}_{k,p}(c,t_o,t_p,i)$
15:     if $i > k$ and $\textsc{Rand}() < p$ then
16:         return $t_p(c)$
17:     else
18:         return $\textsc{Next}(c,t_o)$
```



$[\text{ send }]_\Sigma \; [\text{ him a flower }]_B \; \overset{\textsc{Left-Arc}}{\Longrightarrow} \; [\;]_\Sigma \; [\text{ him a flower }]_B$

Figure 3.10. Cost of left-arc in a given configuration. A blue thick arc is an arc that has been constructed and red dashed arcs are arcs that we can no longer construct.

configuration, we need to define a function which computes the cost of any transition. The cost is defined as the number of gold arcs that are not possible to construct after taking a transition but that were possible to construct before. For example, if after configuration 1 from Figure 3.9, depicted in the left part of Figure 3.10, we do a LEFT-ARC, attaching *send* to *him*, we pop *send* from the stack and therefore lose both the possibility to attach *him* to *send* and *flower* to *send*, as depicted in the right part of the figure. We can still construct the last remaining arc, attaching *a* to *flower*. The cost of LEFT-ARC in configuration 2 is therefore two. More generally, performing a LEFT-ARC transition means that s_0 can no longer be attached to s_1 or any item of the buffer and it will not be able to acquire dependents from the buffer. The cost for a LEFT-ARC transition is therefore the number of dependents of s_0 that are in the buffer plus one if the head of s_0 is s_1 or is in the buffer.

More formally, the cost functions of the different transitions for arc-hybrid as defined in Goldberg and Nivre (2013) are given below:

- $C(\textsc{Left}; c, T)$: Adding the arc (b, s_0) and popping s_0 from the stack means that s_0 will not be able to acquire a head from $H = \{s_1\} \cup \beta$ and will not be able to acquire dependents from $D = \{b\} \cup \beta$. The cost is therefore the number of arcs in T of the form (s_0, d) and (h, s_0) for $h \in H$ and $d \in D$.
- $C(\textsc{Right}; c, T)$: Adding the arc (s_1, s_0) and popping s_0 from the stack means that s_0 will not be able to acquire heads or dependents from $B = \{b\} \cup \beta$. The cost is therefore the number of arcs in T of the form (s_0, d) and (h, s_0) for $h, d \in B$.
- $C(\textsc{Shift}; c, T)$: Pushing b onto the stack means that b will not be able to acquire heads from $H = \{s_1\} \cup \sigma$, and will not be able to acquire dependents from $D = \{s_0, s_1\} \cup \sigma$. The cost is therefore the number of arcs in T of the form (b, d) and (h, b) for $h \in H$ and $d \in D$.

An arc that is not possible to construct is not *reachable* in the terminology of Goldberg and Nivre (2012). If all arcs in the gold tree that were reachable before taking a transition t are still reachable after taking that transition, the cost of t is zero and the oracle considers it an optimal transition. If we take a non-zero cost action (that is, if we follow a non-optimal transition in EXPLORE), the gold tree will no longer be reachable. The cost function then determines whether or not we can still reach the best possible tree, given the arcs that are no longer reachable. An optimal transition is therefore defined as a transition that has a cost of zero. Goldberg and Nivre (2013) have shown that we can easily calculate the cost of every possible transition if the transition system is arc-decomposable. A transition system is arc-decomposable if we can determine for any (projective) tree and for every configuration that the system may be in whether the tree is reachable or not by determining for every arc of the tree whether or not the arc is reachable. In other words, in an arc-decomposable system, we can reduce the reachability of the tree to the reachability of its arcs. Goldberg and Nivre (2013) show that the arc-hybrid system, among other systems but not including the popular arc-standard system, is arc-decomposable which is why Kiperwasser and Goldberg (2016b) selected it as a transition system and we keep it in our extension to their parser.

3.3.3 Combining Reordering and Dynamic Oracles

Transition systems that allow non-projective trees are in general not arc-decomposable and therefore require different methods for constructing dynamic oracles (Gómez-Rodríguez and Fernández-González (2015) show this for the Covington system for example). Consider our transition sequence in Figure 3.8.

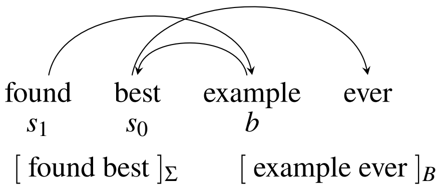


Figure 3.11. Configuration where the cost of shift is 0 if a swap is guaranteed to follow.

With the cost function defined above, the transition Shift in the configuration from step 3, depicted in Figure 3.11, would have a cost of 2: we could no longer attach *best* to *example* and *example* to *found* (because *best* blocks a Right-Arc transition between the two). However, as we can see in the transition sequence in the figure, since we can swap that word back to the buffer in a later step, there is still a chance to construct these arcs. However, if we do not do a Swap transition in step 4, we do in fact lose the possibility to construct these arcs. The cost can therefore not exactly be computed for every transition using just reachability, since the cost of a transition can depend on later transitions.

This problem is partially addressed by Björkelund and Nivre (2015) who propose a non-deterministic oracle for parsing with reordering for arc-standard. Their solution involves a search over possible transition sequences and the oracle complexity is therefore increased to $O(n^2)$ but they find empirically that this makes little difference in training time. Their oracle is non-deterministic, and allows treating different transition sequences as correct, but it is not dynamic and does not allow error exploration.

Another possibility is to learn approximate oracles with reinforcement learning as is done by Yu et al. (2018) or to bypass the use of an oracle by using reinforcement learning to learn rewards (the correct arcs) directly, as in Lê and Fokkens (2017).

We propose to alleviate this problem with a different partial solution: by building a system where the oracle is static with respect to Swap but dynamic with respect to the other transitions. This means that we only make Swap transitions when necessary to make non-projective arcs reachable and we enforce such transitions. We conjecture that this allows us to retain arc-decomposition but we leave proving this conjecture to future work.[8] The cost function for the projective case crucially relies on information about the position of the words in the stack and buffer to compute the cost of a transition. However, once we allow moving items from the stack back to the buffer, we can no longer rely

[8]Our conjecture is based on the observation that we can keep arc-decomposability by decoupling the transition sequence into two passes: the first pass is used for reordering the sequence (using Shift and Swap operations) and the other for constructing the tree. We can deterministically do the first pass if we use a static oracle for Swap and we use the original arc-hybrid system for the second pass, which is arc-decomposable. It remains to be proven that we do not break this property when we interleave the two passes.

on this information. For this reason, we explicitly keep track of the arcs that are still reachable or not: for every word, we keep a list of its reachable dependents, which is initially all of the dependents of the words. We update this list when we take a transition that makes one of these arcs unreachable. We also need to keep track of the projective order of words in the sentence. We therefore assume that gold trees are preprocessed at training time to compute the following information for each input node *i*:

- $\textsc{proj}(i)$ = the position of node *i* in the projective order.
- $\textsc{rdeps}(i)$ = the set of reachable dependents of *i*, initially all dependents of *i*.

Static Oracle for Swap

As described, our oracle is fully dynamic with respect to Shift, Left and Right but basically static with respect to Swap. This means that only optimal (zero cost) Swap transitions are allowed during training and that we force the parser to apply the Swap transition when needed.

Optimal: To prevent non-optimal Swap transitions, we add a precondition so that Swap is legal only if $\textsc{proj}(s_0) > \textsc{proj}(b)$.

Forced: To force necessary Swap transitions, we bypass the oracle whenever $\textsc{proj}(s_0) > \textsc{proj}(b)$.[9]

Dynamic Oracle

Since we use a static oracle for Swap transitions, these will always have zero cost. The dynamic oracle thus only needs to define costs for the remaining three transitions. To construct the oracle, we start from the old dynamic oracle for the projective system and extend it to account for the added flexibility introduced by Swap. Figure 3.12 defines the transition costs as well as the necessary updates to rdeps after applying a transition. The cost functions are defined as such:

- Left: Adding the arc (b, s_0) and popping s_0 from the stack means that s_0 will not be able to acquire a head different from b nor any of its reachable dependents. In the old projective case, the loss was limited to a head in $s_1|\beta$ and dependents in $b|\beta$, but because s_0 can potentially be swapped back to the buffer, we instead define reachability explicitly through $\textsc{rdeps}(s_0)$ (for dependents) and $\textsc{rdeps}(h(s_0))$ (for the head) and update these accordingly after applying the transition.
- Right: Adding the arc (s_1, s_0) and popping s_0 from the stack means that s_0 will not be able to acquire a head different from s_1 nor any of its reachable dependents. In the old projective case, the loss was limited to a head and dependents in $b|\beta$, but because s_0 can potentially be swapped back to

[9]This is equivalent to an *eager* static oracle for Swap in the sense of Nivre et al. (2009).

- Left:

$$\mathscr{C}(\text{Left}) = |\textsc{rdeps}(s_0)| + [\![h(s_0) \neq b \text{ and } s_0 \in \textsc{rdeps}(h(s_0))]\!]$$

  Updates: Set $\textsc{rdeps}(s_0) = [\,]$ and remove s_0 from $\textsc{rdeps}(h(s_0))$.

- Right:

$$\mathscr{C}(\text{Right}) = |\textsc{rdeps}(s_0)| + [\![h(s_0) \neq s_1 \text{ and } s_0 \in \textsc{rdeps}(h(s_0))]\!]$$

  Updates: Set $\textsc{rdeps}(s_0) = [\,]$ and remove s_0 from $\textsc{rdeps}(h(s_0))$.

- Shift:
  1. If there exists a node $i \in B_{-b}$ such that $b < i$ and $\textsc{proj}(b) > \textsc{proj}(i)$:

$$\mathscr{C}(\text{Shift}) = 0$$

  2. Else:

$$\mathscr{C}(\text{Shift}) = |\{d \in \textsc{rdeps}(b) \,|\, d \in \Sigma\}| + [\![h(b) \in \Sigma_{-s_0} \text{ and } b \in \textsc{rdeps}(h(b))]\!]$$

     Updates: Remove b from $\textsc{rdeps}(h(b))$ if $h(b) \in \Sigma_{-s_0}$ and remove $d \in \Sigma$ from $\textsc{rdeps}(b)$.

Figure 3.12. Transition costs and updates. Expressions of the form $[\![\Phi]\!]$ evaluate to 1 if Φ is true, 0 otherwise. We use s_0 and s_1 to refer to the top and second top item of the stack respectively and we use b to denote the first item of the buffer. Σ refers to the stack and Σ_{-s_0} to the stack excluding s_0 (if Σ is not empty). B refers to the buffer and B_{-b} to the buffer excluding b.

the buffer, we again define reachability explicitly through $\textsc{rdeps}(s_0)$ (for dependents) and $\textsc{rdeps}(h(s_0))$ (for the head) and update these accordingly after applying the transition.

- Shift: In the projective case, shifting b onto the stack means that b will not be able to acquire a head in Σ other than the top item s_0 nor any dependents in Σ. With the Swap transition and a static oracle, we also have to consider the case where b can later be swapped back to the buffer, in which case Shift has zero cost. We therefore have two cases in Figure 3.12. In the first case, no updates are needed. In the second case, the updates are analogous to the old projective case.

Evaluation

We integrate this system into our parser, described in the previous section. We first compare our system, which uses our static-dynamic oracle, with the same system using a static oracle. This is to find out if we can benefit from error exploration using our partially dynamic oracle. We additionally compare our method to pseudo-projective parsing. We use the same hyperparameters for the two systems to keep them comparable and just reuse the ones we used for the CoNLL shared task 2017, which will be presented in Section 3.4. Note, however, that they were tweaked for the pseudo-projective system, possibly giving it an unfair advantage. Lastly, we compare to a *projective* baseline,

Table 3.2. *LAS on dev sets with gold tokenization for our static-dynamic system (S-Dy), the static and projective baselines (Static, Proj) and the pseudo-projective system of Section 3.2 (PProj). %NP = percentage of non-projective arcs/trees.*

Language	%NP	S-Dy	Static	PProj	Proj
A.Greek	9.8 / 63.2	**59.5**	56.0	59.2	47.0
Arabic	0.3 / 8.2	**77.1**	76.6	77.0	76.5
Basque	5.0 / 33.7	72.3	71.0	**74.2**	68.8
English	0.5 / 5.0	82.0	81.0	82.2	**82.4**
Portuguese	1.3 / 18.4	**87.3**	86.6	87.2	85.4

using a dynamic oracle but no Swap transition.[10] This is to find out the extent to which dealing with non-projectivity is important.

We select a sample of 5 treebanks from UD version 2.0. (Nivre et al., 2017a) Our main criterion for selecting languages is different from our criteria in Section 3.1. Since the main variable of interest is non-projectivity, we want to have different frequencies of non-projectivity, both at the sentence level and at the level of individual arcs, ranging from a very high frequency (Ancient-Greek) to a low frequency (English). This is to find out the extent to which these different methods affect languages with different ratios of non-projectivity. We still want to ensure to have some typological variety, following the arguments from Section 3.1. Our selection therefore contains Ancient Greek (PROIEL), Arabic, Basque, English and Portuguese. Non-projective frequencies are included in Table 3.2, which report our results on the development sets (best out of 20 epochs).

Comparing to the static baseline, we can verify that our static-dynamic oracle really preserves the benefits of training with error exploration, with improvements ranging from 0.5 to 3.5 points. All differences here are statistically significant with $p < 0.01$, except for Portuguese, where the difference is statistically significant with $p < 0.05$ according to the McNemar test.

The new system achieves results largely on par with the pseudo-projective parser. Our method is better by a small margin for 3 out of 5 languages and worse by a large margin for 1. Overall, these results are encouraging given that our method is simpler and more efficient to train, with no need for pre- or post-processing and no extension of the dependency label set.[11]

Comparing to the projective baseline, we see that strictly projective parsing can be slightly better than both online reordering and pseudo-projective parsing for a language with few non-projective arcs/trees like English. For all other

[10]When training the projective baseline, we removed non-projective trees from the training data, as was done in Kiperwasser and Goldberg (2016b). Note that this means that this baseline therefore has the disadvantage of using less data.

[11]We made no systematic study of training time but observed that it took roughly half the time to train our parser compared to the pseudo-projective one, excluding the pre- and post-processing steps.

languages, we see small (Arabic) to big (Ancient Greek) improvements from dealing with non-projectivity in some way.

This system could also be compared to the non-deterministic oracle from Björkelund and Nivre (2015), and maybe the two approaches can be combined. We leave this to future work.

Overall, we have found a convenient way to deal with non-projectivity in transition-based parsing, answering **RQ1c**: *How can we deal with non-projectivity in transition-based parsing in a convenient way?*

This system was used in Smith et al. (2018a) for the CoNLL 2018 shared task which provided more extensive evaluation. We discuss those results in Chapter 6 where we evaluate the benefits of polyglot parsing, and briefly in Section 3.4.

3.4 Evaluation: the CoNLL Shared Tasks

This section presents a more extensive evaluation of our parser. We participated in the CoNLL 2017 and 2018 shared tasks for which I describe our submissions in turn.

3.4.1 CoNLL Shared Task 2017

We used the K&G parser with some of the modifications that were described above: the use of pseudo-projective parsing and the use of character representations. We call our parser UUParser. UUParser was part of the Uppsala system in the CoNLL 2017 shared task (Zeman et al., 2017). The shared task consisted in predicting dependencies from raw text for 81 test sets in UD. A first modification to the original parser is the replacement of POS tag vectors with character vectors, as described in Section 3.2. We made two additional changes to the original system by Kiperwasser and Goldberg (2016b). First, in order to conform to the constraints of UD representations, we added a new precondition to the Left-Arc_d transition to ensure that the special root node has exactly one dependent. Thus, if the potential head i is the root node, Left-Arc_d is only permissible if the stack contains exactly one element (in which case the transition will lead to a terminal configuration). This precondition is applied only at parsing time and not during training. Second, we made use of pseudo-projective parsing (Nivre and Nilsson, 2005) to allow the construction of non-projective arcs. Pseudo-projective parsing was described in Section 3.3.1. More specifically, we used the pseudo-projective algorithms implemented in MaltParser (Nivre et al., 2006) and we used the *head* schema which I described in Section 3.3.1. This method increases the size of the dependency label set. In order to keep training efficient, we capped the number of dependency relations to the 200 most frequently occurring ones in the training set.

Table 3.3. *Hyperparameter values for parsing.*

Internal word embedding dimension	100
Pre-trained word embedding dimension	50
Character embedding dimension	12
Character BiLSTM dimensions	100
Hidden units in MLP	100
Word BiLSTM Layers	2
Word BiLSTM hidden/output dimension	250
Word BiLSTM dropout	0.33
α (for word dropout)	0.25
Character dropout	0.33
p_{agg} (for exploration training)	0.1

This parser is one of the two main components of the pipeline of the Uppsala system. The first component of the pipeline is a model for joint sentence and word segmentation, which uses the BiRNN-CRF framework of Shao et al. (2017, 2018) to predict sentence and word boundaries in the raw input and simultaneously marks multiword tokens that need non-segmental analysis. The latter are handled by a simple dictionary lookup or by an encoder-decoder network. We used a single universal model regardless of writing system, but trained separate models for each language. See Section 2 of de Lhoneux et al. (2017a) for more details. We constructed word embeddings based on the RSV model of Basirat and Nivre (2017), using universal part-of-speech tags as contexts, see Section 4 of de Lhoneux et al. (2017a) for more details. Our word vectors are thus constructed as such: each word x_i is represented by a randomly initialised embedding of the word $e(w_i)$, a pre-trained embedding $pe(w_i)$ and a character vector $ce(w_i)$:

$$x_i = [e(w_i); pe(w_i); ce(w_i)] \tag{3.3}$$

We first worked with these vectors with the idea in mind to build a single parser for all languages. Due to practical and time constraints, we mostly trained monoligual parsers instead. We did use a multilingual parser for some of the languages which were very low-resource with a method described in Chapter 6. We used the set of hyperparameters reported in Table 3.3.[12] Note that we apply dropout to the word embeddings at the input of the BiLSTM with a probability proportional to the frequency of the word, as described in Kiperwasser and Goldberg (2016b), as well as to the word BiLSTM.

[12]Note that the dropout and exploration parameters are constant throughout the experiments in the thesis, and are therefore only reported here.

Table 3.4. *Results for LAS F1, sentence and word segmentation. δ = diff to Baseline. Scores that are significantly higher/lower than the baseline are marked with two shades of green/red, with brighter colours for larger differences.*

Language	Treebank	LAS F1	δ	Sents	δ	Words	δ
Ancient Greek		58.83	2.79	98.93	0.50	99.98	0.03
Ancient Greek	PROIEL	69.04	3.82	48.86	5.75	99.98	-0.02
Arabic		68.68	3.38	78.21	-6.36	94.99	1.30
Arabic	PUD	50.70	7.56	98.66	-1.34	95.30	4.48
Basque		73.82	4.67	100.00	0.42	100.00	0.04
Bulgarian		85.38	1.74	95.23	2.40	99.91	0.00
Buryat		18.01	-13.49	87.37	-4.44	97.71	-1.64
Catalan		87.08	1.69	99.59	0.64	99.79	-0.18
Chinese		65.25	7.85	98.80	0.61	93.43	4.52
Croatian		79.51	2.33	97.25	0.33	99.91	-0.02
Czech		86.83	3.96	92.79	0.76	99.98	0.08
Czech	CAC	85.75	3.29	99.68	-0.32	99.99	0.00
Czech	CLTT	75.67	4.03	96.95	1.89	99.78	0.43
Czech	PUD	82.27	2.47	95.55	-0.88	99.25	-0.04
Danish		77.70	4.32	83.41	4.05	100.00	0.31
Dutch		74.41	5.51	76.16	-0.98	99.85	-0.03
Dutch	LassySmall	83.58	5.43	87.00	8.38	99.97	0.04
English		79.62	3.78	80.26	7.04	99.05	0.38
English	LinES	75.80	2.86	87.17	1.33	99.96	0.02
English	ParTUT	76.11	2.47	98.10	0.59	99.54	0.05
English	PUD	80.49	1.54	96.15	-0.98	99.59	-0.07
Estonian		58.67	-0.12	93.23	8.03	99.90	0.13
Finnish		78.41	4.66	91.48	6.92	99.71	0.08
Finnish	FTB	76.25	2.22	87.16	3.33	99.99	0.11
Finnish	PUD	80.05	1.40	91.64	-2.03	99.59	-0.02
French		83.66	2.91	94.32	0.73	99.53	0.66
French	ParTUT	80.84	3.46	99.50	1.50	99.50	0.55
French	PUD	75.25	1.62	91.33	-0.99	97.34	-0.83
French	Sequoia	82.85	2.87	84.95	1.20	99.48	0.42
Galician		79.01	1.70	96.83	0.68	99.96	0.04
Galician	TreeGal	65.85	0.03	83.79	2.16	98.23	-0.39
German		75.27	6.16	81.47	2.36	99.67	0.02
German	PUD	70.90	4.37	86.83	0.34	96.43	-1.57
Greek		80.46	1.20	91.09	0.30	99.80	-0.08
Gothic		62.62	2.81	35.01	7.16	100.00	0.00
Hebrew		67.75	10.52	99.69	0.30	91.19	6.37
Hindi		89.13	2.36	99.11	-0.09	99.99	-0.01
Hindi	PUD	53.31	2.46	95.00	4.17	98.65	0.84
Hungarian		65.90	1.60	97.65	3.80	99.89	0.07
Indonesian		76.13	1.52	93.55	2.40	100.00	0.01
Irish		63.35	1.83	95.35	-0.46	99.78	0.49
Italian		87.33	2.05	99.38	2.28	99.86	0.13
Italian	PUD	85.59	1.89	93.90	-2.68	99.28	0.11
Japanese		81.54	9.33	94.92	0.00	93.59	3.91
Japanese	PUD	83.26	6.98	97.31	2.42	94.30	3.24
Kazakh		17.14	-7.37	86.26	4.88	96.46	1.55
Korean		74.72	15.63	93.01	-0.04	99.99	0.26
Kurmanji		20.39	-11.96	94.08	-2.94	97.06	-1.79
Latin		46.26	2.49	97.41	-0.68	100.00	0.01
Latin	ITTB	82.34	5.36	92.93	-0.31	99.99	0.00
Latin	PROIEL	63.17	5.63	34.66	8.86	100.00	0.00
Latvian		59.75	-0.20	93.65	-4.94	99.13	0.22
North Sami		11.72	-18.88	97.59	-1.20	96.75	-3.13
Norwegian	Bokmaal	86.04	2.77	96.20	0.44	99.87	0.12
Norwegian	Nynorsk	84.41	2.85	93.67	2.44	99.92	0.07
Old Church Slavonic		67.04	4.28	43.72	7.67	99.94	-0.02
Persian		81.89	2.65	99.50	1.50	99.61	-0.03
Polish		82.33	3.55	99.46	0.55	99.93	0.05
Portuguese		83.25	1.14	90.43	0.64	99.45	-0.07
Portuguese	BR	88.19	2.83	97.04	0.20	99.87	0.03
Portuguese	PUD	74.48	0.52	94.50	-1.15	98.48	-0.94
Romanian		81.68	1.80	96.02	2.60	99.75	0.11
Russian		77.99	3.96	96.91	0.49	99.90	-0.01
Russian	PUD	70.78	2.47	98.80	-0.15	97.34	0.16
Russian	SynTagRus	89.61	2.85	98.78	0.97	99.63	0.06
Slovak		75.98	3.23	87.17	3.64	99.96	-0.04
Slovenian		84.16	3.01	98.11	-1.13	99.97	0.01
Slovenian	SST	46.76	0.31	19.52	2.80	100.00	0.18
Spanish		84.26	2.79	94.16	0.01	99.84	0.15
Spanish	AnCora	86.79	3.01	98.46	1.41	99.92	-0.03
Spanish	PUD	79.01	1.36	93.39	-0.03	99.34	-0.13
Swedish		79.86	3.13	95.96	-0.41	99.77	-0.07
Swedish	LinES	76.37	2.08	88.12	1.68	99.99	0.01
Swedish	PUD	69.52	-1.10	81.14	-9.06	98.47	0.21
Turkish		52.84	-0.35	96.44	-0.19	97.51	-0.38
Turkish	PUD	32.84	-1.69	90.75	-3.16	96.93	0.31
Ukrainian		59.33	-1.43	92.18	-0.41	99.52	-0.29
Upper Sorbian		47.92	-5.91	89.88	-0.81	98.76	-1.08
Urdu		79.31	2.62	98.60	0.28	100.00	0.00
Uyghur		30.98	-3.20	69.36	5.81	98.74	0.22
Vietnamese		42.68	5.21	89.49	-3.10	86.70	4.23
Average		70.49	2.14	89.48	0.99	98.79	0.30

Our post-evaluation results are reported in Table 3.4.[13] Our average LAS score of 70.41 would put us in the top 10 of the 33 participating teams and 6 LAS points on average below the winning system, Stanford, which uses the Dozat and Manning (2017) system mentioned in Chapter 2. Given that our system is extremely simple in that we do not predict anything other than token boundaries and dependency relations, we find these results very encouraging. The systems that outperformed ours either used a graph-based parser, for which there is an established way to allow the construction of non-projective trees, or used an ensemble system, which boosts accuracy substantially. Che et al. (2017), for example, report a 1 LAS point improvement on several development sets when using an ensemble.

Our results are compared to the shared task baseline in Table 3.4, UDPipe (Straka et al., 2016). To make the table somewhat more readable, we added a simple colour coding. Scores that are significantly higher/lower than the baseline are marked with two shades of green/red, with brighter colours for larger differences. Thresholds have been set to 1 and 3 points for LAS, 0.5 and 1 points for Sentences, and 0.1 and 0.5 points for Words. We observe that our results are substantially worse than the baseline only for the languages that are very low resource. These languages are the ones for which we attempted to build a multilingual model. This may give some indication that replacing the POS tag with a character vector is not beneficial when using cross-lingual parsing in the very low-resource setting. A confounding factor, however, is that the predicted POS tags for the baseline system were obtained by doing jackknifing on the test set, giving it an unfair advantage. This makes it difficult to properly compare the two systems and our conclusion that character vectors are not beneficial in cross-lingual parsing in the very low-resource setting needs more investigation. For the rest, our results are, for the most part, substantially above the baseline. We observe that our results are especially good for languages which have a different script than the Latin script, for example Arabic, Chinese, Hebrew, Japanese, Korean and Vietnamese. For those languages (except Korean), our segmentation results are, on average and for most languages, substantially better than the baseline, which can partially explain why our parsing results are also better. More interestingly, we observe that our results are especially good for languages that are at least somewhat morphologically complex like Ancient Greek, Arabic, Basque, Czech, Finnish, German, Latin, Polish, Russian and Slovenian. This gives further evidence that char-

[13]We found two bugs after the official evaluation. The official results can be found in de Lhoneux et al. (2017a). Concerning the parser, our official test run suffered from a bug in the way serialization is handled in all dynet versions preceding dynet 2.0. As reported in https://github.com/clab/dynet/issues/84, results may differ if the machine on which a model is used does not have the exact same version of the boost library as the machine on which the model was trained. Our post-evaluation results were obtained by using exactly the same models but parsing the test data on the machine on which they were trained. There was a bug in the segmenter as well which is explained in our paper.

Table 3.5. *Hyperparameter values for parsing.*

Character embedding dimension	500
Character BiLSTM output dimension	200
Word embedding dimension	100
POS embedding dimension	20
Treebank embedding dimension	12
Word BiLSTM layers	2
Word BiLSTM hidden/output dimension	250
Hidden units in MLP	100

acter models can 1) compensate for the lack of POS tags and 2) compensate for the lack of explicit morphological features. It seems that a model which replaces POS tags with character vectors is a strong baseline parser which is robustly good across languages, compared to previous methods.

3.4.2 CoNLL Shared Task 2018

In the CoNLL 2018 shared task, we used the reordering method and the static-dynamic oracle presented in Section 3.3.1. We also re-introduced the use of POS tags, used pre-trained embeddings and polyglot parsing. Our method for polyglot parsing will be explained in Chapter 6. We used the pre-trained embeddings provided by the organisers of the CoNLL 2017 shared task, as well as Facebook word embeddings (Bojanowski et al., 2017) for treebanks for which those pre-trained embeddings are not provided, and both word and character embeddings trained on Wikipedia text with word2vec (Mikolov et al., 2013b). Unlike in the CoNLL 2017 shared task, we did not use randomly initialised embeddings of the word types since we found in preliminary experiments that using only pre-trained embeddings was just as good and more efficient. We used the hyperparameters from Smith et al. (2018b) where we found, among other things, that large character embedding sizes are beneficial. The hyperparameters are listed in Table 3.5.

Similarly to Section 3.4.1, our parsing model is part of a pipeline, consisting of three components here. The first component for sentence segmentation and tokenization is the same as the one described in Section 3.4.2. The second component is a part-of-speech (POS) tagger based on Bohnet et al. (2018), which employs a sentence-based character model and also predicts morphological features. See Section 4 in Smith et al. (2018a) for more details. Our parser is the third and last component. Our system ranked 7th in the task and was the best transition-based parser. Our results were on average only 2.5 LAS points below the best system and on par with the best systems which do not make use of contextualised representations which boost accuracy substantially. The paper of the winning system by Che et al. (2018) reports a 0.84 LAS points improvement from using ELMo, the contextual representations developed by Peters et al. (2018), compared to not using ELMo on the large treebanks. We

Table 3.6. *CoNLL 2018 shared task ranked average LAS on a representative sample. The last column represents the movement compared to the original ranking.*

rank	system	LAS	mov.
1	HIT-SCIR-18	73.1	=
2	Stanford-18	71.9	+5
3	Uppsala-18	70.7	+4
4	TurkuNLP-18	69.9	−2
4	LATTICE-18	69.8	−1
4	UDPipeFuture	69.7	−1
4	CEA LIST	69.4	+2
4	ICS PAS	69.3	−1
9	NLP-Cube	68.8	=
9	ParisNLP	68.3	+2
11	AntNLP	67.3	−2
11	LeisureX	67.0	+4
13	UniMelb	66.3	+1
13	SLT-Interactions	66.0	−1
13	IBM NY	65.9	=
16	BASELINE	63.7	+2
16	Phoenix	63.2	+3
16	KParse	62.9	=
19	Fudan	61.7	−2
19	CUNI x-ling	61.3	+1
21	BOUN	60.6	=
22	ONLP lab	56.3	=
23	ArmParser	54.4	+2
24	HUJI	46.4	=

obtained the best word and sentence segmentation results as well as the best POS tagging accuracy, which do give us an advantage over the other systems at parsing time, however.

As argued in this chapter, a problem with evaluating parsers on the full set of treebanks is that it might give an advantage to systems that are biased towards, for example, Indo-European languages. As argued in Section 3.1, evaluating on a representative sample mitigates this problem. We evaluate the shared task outputs on the sample proposed in Section 3.1, excluding Tamil which was not part of the shared task, for all systems that have produced an output for each of those test sets, including the baseline system. Our selection is therefore: Czech (PDT), Chinese (GSD), Finnish (TDT), English (EWT), Ancient-Greek (PROIEL), Kazakh (KTB) and Hebrew (HTB). I report the ranked results in Table 3.6 and include the movement compared to the original ranking.

The ranking does not substantially change: the large majority (20 out of 24) of systems moved 2 positions maximum. This can either indicate that all systems are biased in a similar way or that evaluating on the full set does not give a

Table 3.7. *Properties of the K&G parser and of UUParser. Parentheses denote options while the rest are default settings.*

	K&G	UUParser
Word embedding	✓	✓
POS embedding	✓	(✓)
Character vector	✗	✓(see Section 3.2)
Treebank embedding	✗	(✓) (see Section 6.2)
Polyglot training	✗	(✓) (see Section 6.2)
Non-projective parsing	✗	✓(see Section 3.3.1)
Dynamic oracle	✓	partial (see Section 3.3.3)

substantial advantage to systems that are more biased than others. Our system, Uppsala-18, is one of the systems with the biggest jumps: we moved 4 positions up to reach the 3d place.[14] This could indicate that our parser is less biased than others. However, there are several problems with this evaluation which strongly limit the conclusions we can reach from this. One problem is that we cannot isolate parsing from segmentation results, and we might be advantaged by our highly accurate segmenter here more than in the global evaluation. Another problem is that we have been using a very similar sample for parser development in several studies. This might also give us an unfair advantage. Ideally, the shared task would have development and evaluation languages, but this is complicated in practice since UD treebanks have been available as open source for a long time. Perhaps more importantly, we cannot isolate the gains obtained from polyglot parsing (see Chapter 6). Our system was 5 LAS points better than the second best system for Kazakh, and 8 LAS points better than our monolingual baseline for Kazakh. This largely overestimates our ranking compared to if we used our monolingual baseline. On the other hand, other systems also use polyglot parsing and similar tricks. We would need to evaluate the monolingual baselines of all systems to draw clearer conclusions from this. Meanwhile, it seems reasonable to conclude that our parser is competitive for typologically diverse languages.

Although this section provides a less direct evaluation of the specific way in which we deal with non-projectivity compared to our experiments in Section 3.3, it indicates that this is a viable approach for transition-based dependency parsing. Our CoNLL 2018 shared task contribution and our results are presented more at length in Chapter 6.

I summarise the properties of UUParser compared to the K&G transition-based parser in Table 3.7. The main drawback of our system is that our oracle is not fully dynamic. For the rest, we have added several options which improve the parser and made the use of POS tags optional so that our default parser is a simple but strong baseline.

[14]Note that the Stanford system moved even more in the ranking but this is likely due to a bug which affected pre-processing big treebanks, as reported by Qi et al. (2018).

3.5 Conclusion

In this chapter, I presented various approaches to building models that are suitable for typologically diverse languages. In Section 3.1, we proposed a general method for working with a dataset containing a large number of typologically diverse languages. We proposed a set of criteria to sample a subset of treebanks from the UD treebanks so that it is representative typologically and we showed how these criteria can be applied. The application of these criteria is done manually but future work could automate this process. We gave arguments as to why it is beneficial to use such a sample for parser development and evaluation, inspired by research in typology and typologically aware NLP. Working with such a sample allows us to be reasonably sure that our results generalise well across languages, or at least are not heavily biased towards the overrepresented languages of UD. This method is used in Chapter 4 where we do a thorough investigation of the use of a tree layer in our parser with various model ablations. It is also used and adapted in Chapter 5 where we do an in-depth analysis of what our parser learns about auxiliary verb constructions. It is finally used and adapted in Chapter 6 where we examine parameter sharing strategies across typologically varied language pairs.

I went on to describe a way to make parsers more robust across languages along the morphological complexity dimension by making use of character models. Character models were shown by Ballesteros et al. (2015) to be more robustly useful than POS tags across a set of languages, and to improve parsing especially for morphologically complex languages. We verified that this finding holds when using our parsing architecture. We use character vectors in our parser in experiments in the remainder of this thesis.

I then described the problem of non-projective parsing together with different solutions. I described the advantage of using dynamic oracles instead of static oracles when training parsers and I showed the difficulty of integrating this method with an efficient method for non-projective parsing: using reordering. We proposed a partial solution to this problem which makes use of an oracle that is static with respect to the reordering operations, the Swap transition, but dynamic with respect to the other transitions. We showed that this method preserves the advantage of training with a dynamic oracle. We provided empirical evidence that it works at least as well as another method which works well but has several disadvantages: pseudo-projective parsing. We use this technique in our experiments in the remainder of this thesis.

Finally, I presented a more extensive evaluation of the parser by presenting our participation in the CoNLL 2017 and 2018 shared tasks where we performed competitively, in spite of having a simple system. The results, both on the full set of test sets and on a representative sample, confirmed that our parser is close to state-of-the-art and the best current transition-based parser on UD data. We have provided answers to **RQ1**: *How can we build parsing models that perform well across typologically diverse languages?*

We release an open source version of UUParser.[15] While we used POS tags for the CoNLL 2018 shared task, we advocate refraining from using them when analysing parsing models, to minimise the number of confounding variables. Our baseline without POS tags is very strong and while using POS tags usually improves parsing accuracy, the gains are not substantial. We advocate refraining from using POS tags even more strongly in polyglot parsing, especially in a scenario where we want to improve parsing for low-resource languages, because obtaining accurate POS tags is not realistic for these languages. I will return to that in Chapter 6. We only use POS tags in Chapter 4 to study their interaction with our tree layer, but do not use them in other experiments in this thesis (besides the CoNLL 2018 shared task).

Similarly, structural features, which were briefly mentioned in Chapter 2 (vector representations of children of elements of the stack and buffer), only provide minor improvements in parsing accuracy.[16] For this reason, we also do not include them in our experiments (with the exception of the shared tasks again), to further minimise the number of confounding factors. Our baseline parser is therefore very simple while staying competitive.

[15]The latest version is at `http://github.com/UppsalaNLP/uuparser`. Our CoNLL 2017 system is released as UUParser 1.0 (`https://github.com/UppsalaNLP/uuparser/releases/tag/1.0`) and our CoNLL 2018 system is released as UUParser 2.3 (`https://github.com/UppsalaNLP/uuparser/releases/tag/2.3`).

[16]Falenska and Kuhn (2019) actually find that they are mostly unhelpful.

4. Recursive Subtree Composition

This chapter is a study about the usefulness of incorporating a hierarchical bias in a parsing model architecture. The aim here is to answer **RQ2**:*How can we incorporate linguistic knowledge into neural models for dependency parsing?* In Chapter 2, I introduced UD and the opportunities it opens up for building models that incorporate linguistic knowledge without tying this knowledge to a specific language. I described ways in which linguistic knowledge can be incorporated into neural NLP models. One of the methods is to use linguistic knowledge to guide neural architecture engineering, which we explore here. I gave the example of Dyer et al. (2015), who showed the impact of using explicit hierarchical composition in their parsing model. I described the line of work by Linzen et al. (2016) which tested the capacity of LSTMs to learn hierarchical structure. This line of study, which I describe in more detail in this chapter, has shown that LSTMs do seem to learn hierarchy which casts some doubt on the necessity of explicitly modelling hierarchical composition. In Chapter 3, I described how we can parse typologically diverse languages. I presented UUParser, our parser which performs competitively on UD treebanks. In this chapter, we re-evaluate the findings of Dyer et al. (2015) using our parser. We work on a larger variety of languages than has been done previously, although we still make use of a sample, as proposed in Chapter 3 in order to make sure that our analyses are not biased towards a specific language family. This also allows us to do detailed analysis.

I first introduce background studies about hierarchical structure and how to probe it in neural networks in more detail than in Chapter 2. I then present experiments which cast some doubt on the usefulness of hierarchical modelling in our parsing model. We do extensive analysis to understand this result. In Chapter 5, we look at a specific case where it might still be useful: for modelling the transfer relation between two words in a dissociated nucleus.

4.1 Hierarchical Structure

This section first describes hierarchical structure from a linguistic and typological perspective. I then describe how we can probe hierarchical biases of neural networks.

4.1.1 Hierarchical Structure in Language

There is a large body of work in linguistics about language being hierarchical or recursive, notably in Chomsky (1957). More recently, Everaert et al. (2015)

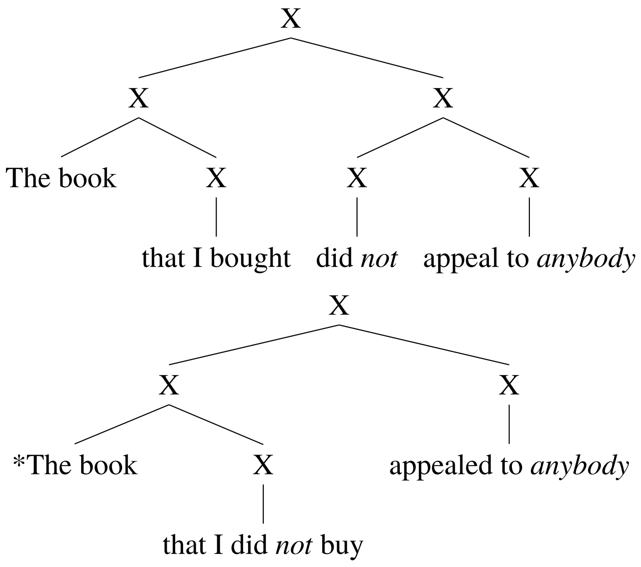


Figure 4.1. Trees with negative polarity.

discuss hierarchical structure by giving the example of *negative polarity items* like *anybody* for English and a related example from Japanese. If we take an example of a sentence with *anybody* such as Example 1a and remove negative polarity in the sentence (Example 1b, where *did not* is removed), or if we move negative polarity to a different place in the sentence (Example 1c where the verb *buy* becomes negative and the verb *appeal* becomes positive), we get an ungrammatical sentence.[1]

(1) a. The book I bought did *not* appeal to *anybody*
    b. *The book I bought appealed to *anybody*
    c. *The book I did *not* buy appealed to *anybody*

The reason Example 1c is not licensed can be explained when we look at the tree structures of Example 1a and 1c. We can see in the bottom tree corresponding to 1c that the word *not* is lower in the tree than the word *anybody*, whereas in the top tree corresponding to 1a, it is in a sibling constituent.[2] Everaert et al. (2015) take this example as evidence that a hierarchical constraint governs this pattern which cannot be explained by a sequential constraint (the constraint that *not* should precede *anybody* is clearly not a constraint that is restrictive enough). They claim that similar constraints appear to hold across languages and in many other syntactic contexts.

[1]Those examples are taken from Dyer (2017) who adapted them from Everaert et al. (2015).
[2]They are in a *c-command* relation, see Reinhart (1981).

There is also a lot of work discussing whether or not hierarchy or recursion in language is universal. Hauser et al. (2002) make the controversial claim that recursion is the only faculty of language that is universal. What is controversial about this claim is both that recursion is universal (Everett (2005) finds that Pirahã, a language of Brazil, lacks recursion) and unique to humans (Gentner et al. (2006) found evidence that songbirds can recognise recursion in human language). Here, we are interested in the question of whether or not hierarchical structure is a useful bias for neural models when working from a typological perspective. Whether this faculty is unique to humans is therefore irrelevant. We are also interested in a less strict claim than that faculty being universal. It does not have to be universal to be a useful bias in a typological perspective. The claim from Everaert et al. (2015) that many languages display hierarchical constraints seems to indicate that this is, indeed, a useful bias.

In Chapter 2, we argued that UD allows us to incorporate linguistic knowledge into our parser that is neither too general to be useful, nor too specific to a language or a restricted set of languages. The notion of hierarchy is a rather general one and its incorporation in a model arguably does not need consistently annotated treebanks. However, the fact that we have a common representation means that we will learn trees that have the same shape for the same phenomena. If it were not the case, this would add a confounding factor. Additionally, as will be described in Section 4.3.2, we use dependency labels in subtree representations. It is therefore also important to work with the same label set for all languages.

4.1.2 Probing Hierarchical Structure

Several studies have set up tasks with the aim of probing the hierarchical biases of neural models. I describe the tasks in this section and discuss results of studies using those tasks in Section 4.2.

Linzen et al. (2016) set up the subject–verb agreement agreement task: in order to know the form of a verb in the English third person in the present tense, you need to know the number of the subject, as can be seen from Example 2.

(2) a. The **kids admire** her.
   b. *The **kid admire** her.
   c. *The **kids admires** her.
   d. The **kid admires** her.

The task may seem trivial if we only look at these examples, since the subject directly precedes the verb. However, if words intervene between the subject and the verb such as in Example 3, it can become more difficult if we only have access to sequential information. In this example, the noun just preceding the verb has a different number and may cause confusion. However, if we have

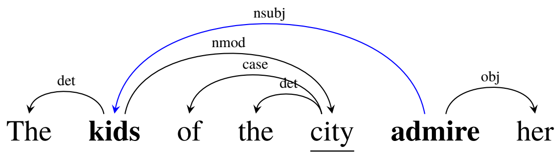


Figure 4.2. Tree with intervening noun.

access to the syntactic structure of the sentence such as the tree in Figure 4.2, the task is trivial since there is a direct *nsubj* link between the verb and its subject. Linzen et al. (2016) therefore propose that this task is an appropriate test bed for probing the hierarchical capacity of a model.

(3) The kids of the city admire her.

They test an LSTM on this task trained with a language modelling objective. When provided with a test sentence where the verb is omitted, if the language model assigns a higher probability to the correct than to the incorrect form, it is considered to be correct for that example. Gulordava et al. (2018) extend this framework and construct probing tasks of long-distance number agreement for 4 languages: Italian, English, Hebrew and Russian. They also extend this framework by creating what they call *nonce sentences* such as in Example 4, where the sentence is grammatical but meaningless. This is to control that LSTMs do not rely exclusively on lexical cues to solve the agreement task.

(4) The colorless green *ideas* I ate with the chair *sleep* furiously.

Ravfogel et al. (2018) look at verb number prediction and suffix recovery in Basque. They show that this is a more challenging benchmark than the English subject–verb agreement as it requires implicit understanding of sentence structure and learning a morphological system. Ravfogel et al. (2019) further extend this framework to systematically study the effect of typological variation on learning such tasks. They construct a subject and object agreement task and test it on synthetic data where word order and morphological marking is systematically altered to control for these variables. Other tasks designed to test the syntactic capacities of models include filler–gap dependencies (Wilcox et al., 2018), grammaticality (Marvin and Linzen, 2018) and long-distance dependencies (Merlo and Ackermann, 2018; Merlo, 2019).

4.2 Recursive vs Recurrent Neural Networks

In Chapter 2, I introduced recurrent neural networks, which are sequence models. A recursive neural network models hierarchical structure instead. The top part of Figure 4.3 depicts a recurrent neural network passed over the words of

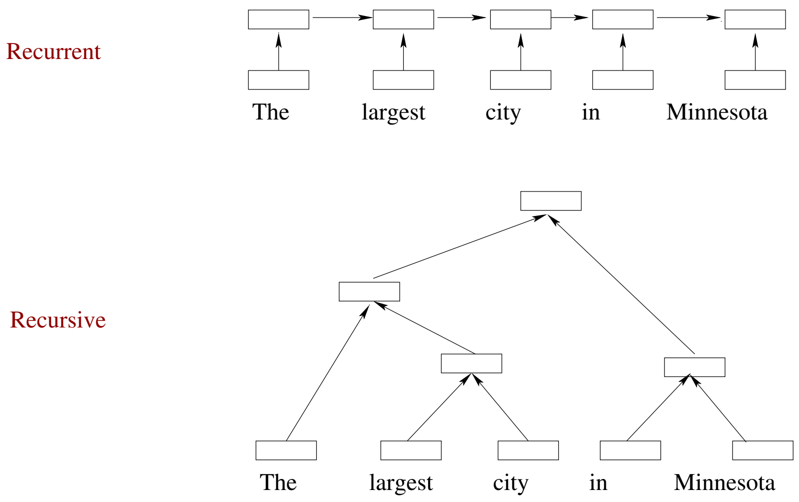


Figure 4.3. Recursive vs recurrent neural network.

a sentence. The bottom part of the figure depicts a recursive neural network: it composes subtree representations recursively.

Recursive neural networks have been used for constituency parsing by Socher et al. (2013) and Dyer et al. (2016) and for dependency parsing by Stenetorp (2013), Dyer et al. (2015) and Kiperwasser and Goldberg (2016a), among others. In particular, as described in Chapter 2, Dyer et al. (2015) showed that composing representations of subtrees using recursive neural networks can be beneficial for transition-based dependency parsing. Dyer et al. (2015) define a recursive composition function and compose tree representations incrementally, as dependents get attached to their head (as explained briefly in Chapter 2 and in more detail in Section 4.3.1). They create two versions of the parser, one which uses this composition function and one which does not. They observe that the version with composition is substantially better than the version without it on the Penn Treebank (PTB) and the Chinese Treebank (CTB), as can be seen in Table 4.1 where we repeat these results. They conclude that this composition function is important. These results are further strengthened in Kuncoro et al. (2017) who showed, using ablation experiments, that composition is key in the Recurrent Neural Network Grammar (RNNG) generative parser by Dyer et al. (2016).

The BiLSTM parser by Kiperwasser and Goldberg (2016b) which has been described at length in this thesis, obtained results on par with the stack-LSTM parser with composition by Dyer et al. (2015) on the PTB and the CTB, see Table 4.1. This raises the question of whether or not recursive composition is useful with a BiLSTM parser or whether BiLSTMs capture hierarchical structure and make the use of a recursive composition function superfluous.

Table 4.1. *LAS scores of a recursive neural network parser (Dyer et al., 2015) and a parser using recurrent neural networks only (Kiperwasser and Goldberg, 2016b).*

	PTB	CTB
Dyer et al. (2015) without composition	89.6	83.6
Dyer et al. (2015) with composition	90.9	85.7
Kiperwasser and Goldberg (2016b) transition-based	91.2	85.0

Using the tasks which probe hierarchical structure described in Section 4.1.2, Linzen et al. (2016) and Gulordava et al. (2018) find that purely sequential LSTMs indeed learn some notion of hierarchy. Linzen et al. (2016) show that LSTMs trained for the task of language modelling do not learn this but they do learn it when trained on the subject–verb agreement task. Gulordava et al. (2018) show that if we give more capacity to an LSTM trained for language modelling, it can actually learn to perform well on agreement tasks. They also show that LSTMs are capable of learning agreement tasks when tested on *nonce* sentences as described above, indicating that they do not entirely rely on lexical cues for this task. They show that LSTMs are capable of learning long-distance number agreement for English, Italian, Hebrew and Russian. In the same vein, Wilcox et al. (2018) find that LSTMs can learn filler–gap dependencies. Tran et al. (2018) find that Transformers cannot learn agreement, indicating that the recurrence of LSTMs is important in learning the task.

Ravfogel et al. (2018) and Ravfogel et al. (2019) cast some doubts on the subject–verb agreement results. Ravfogel et al. (2018) show that the case of agreement in Basque is more difficult than in English. Experiments using diagnostic classifiers suggest that the network uses local heuristics as a proxy for hierarchical structure. Ravfogel et al. (2019) show that the agreement task is easier for SVO languages than SOV languages. Their results indicate that LSTMs have a recency bias. Marvin and Linzen (2018) find that LSTMs, even when supervised with syntactic information, are far from human performance when it comes to grammaticality judgements, suggesting that the syntactic capacity of LSTMs has room for improvement. Merlo (2019) further finds that word and sentence embeddings obtained from RNNs do not encode linguistic properties of long distance dependencies. Kuncoro et al. (2018), in addition, show that recursive LSTMs are better at the English subject–verb agreement task than sequential LSTMs. It seems from all these results that 1) LSTMs are far from perfect at capturing syntactic information and 2) a recursive neural network is a better model of syntactic structure than a purely sequential model. Adding a tree layer on top of the BiLSTM parser from Kiperwasser and Goldberg (2016b) could therefore be beneficial and help improve parsing accuracy which is what we test in this chapter. We observe that it does not improve accuracy and we investigate the interaction between the recursive layer and other parts of the network, through various model ablations.

We therefore ask the following questions:

RQ2a Can a BiLSTM parser benefit from recursive subtree composition?
RQ2b How does composition interact with different parts of the network?

4.3 Methodology

In this study, we 1) re-evaluate the finding of Dyer et al. (2015) that recursive composition is crucial for parsing. We test this claim on more languages and using our simpler parsing architecture 2) we investigate the interaction between recursive composition and other parts of our network by doing model ablations. I first start by explaining in detail the use of a recursive composition function in the stack-LSTM parser (S-LSTM). I use the term *composition* to refer to a recursive composition function in the remainder of this thesis.

4.3.1 Composition in S-LSTM

As mentioned in Chapter 2, the S-LSTM has a complex architecture. It uses a variant of the LSTM called a stack LSTM. A stack LSTM has push and pop operations which allow passing through states in a tree structure rather than sequentially. Stack LSTMs are used to represent the stack, the buffer, and the sequence of past parsing actions performed for a configuration. In the initial configuration, vector representations of all words of the sentence are in the buffer and the stack is empty. The representation of the buffer is the end state of a backward LSTM over the word vectors. As parsing evolves, the word vectors are popped from the buffer, pushed to and popped from the stack and the representations of stack and buffer get updated. This parsing architecture is represented in Figure 4.4. A configuration or parser state is represented by p_t which is a linear transformation of the LSTM representations of the stack, buffer and sequence of past actions. p_t is then passed through an affine transformation and a softmax layer to score possible parsing actions.

The words of the sentence are represented by vectors of the word types, together with a vector representing the word's POS tag. More precisely, the vector x_i of a word w_i is the concatenation of a randomly initialised word embedding $e(w_i)$, a pretrained word embedding $pe(w_i)$ and an embedding of the word's POS tag $e(t_i)$. This concatenated vector is passed through an affine transformation and a ReLU activation as in Equation 4.1.

$$x_i = max\{0, W([e(w_i); pe(w_i); e(t_i)]) + b\} \tag{4.1}$$

In the initial configuration, the vectors x_i of all words are in the buffer and the stack is empty. The representation of the buffer is the end state of a backward LSTM over the word vectors. As parsing evolves, the word vectors are popped

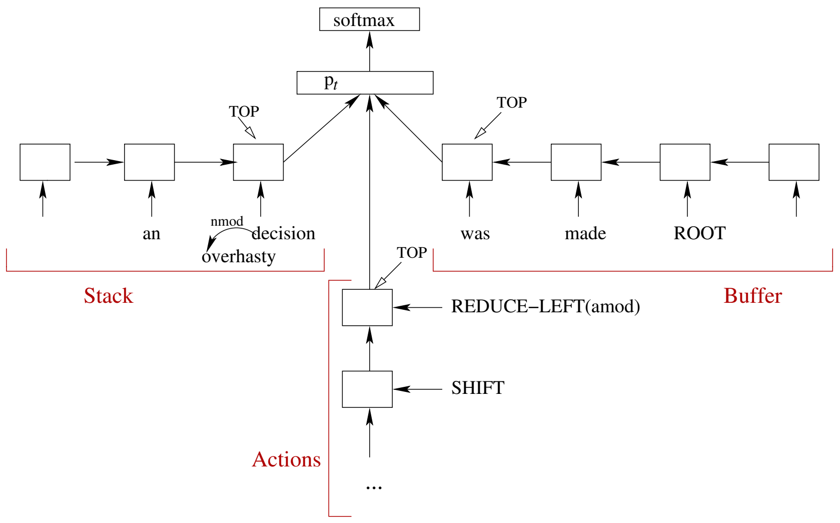


Figure 4.4. Stack LSTM parsing architecture for a given configuration.

from the buffer, pushed to and popped from the stack and the representations of stack and buffer get updated. Figure 4.4 represents a configuration where three words remain in the buffer and two in the stack.

As mentioned in Section 2.4.1, Dyer et al. (2015) define a recursive composition function and compose tree representations incrementally, as dependents get attached to their head. The composed representation c is built by concatenating the vector h of the head with the vector of the dependent d, as well as a vector r representing the label paired with the direction of the arc. That concatenated vector is passed through an affine transformation and then through a *tanh* non-linear activation. The vector h is then replaced by the output of that composition function, c. In Figure 4.4, *overhasty* has just been attached to *decision* and the representation x_i of the word *decision* gets updated with this recursive composition function. Dyer et al. (2015) find that this composition function is highly beneficial: a parser that uses it is substantially more accurate than a parser that does not (i.e. a parser that does not modify the representation x_i of a head word when attaching a dependent to it), it performs more than 1 LAS points better for the PTB, more than 2 LAS points better for the CTB (see Table 4.1).

$$c = tanh(W[h;d;r]+b) \tag{4.2}$$

4.3.2 Composition in UUParser

Our parser, like the S-LSTM, builds trees bottom-up, attaching words one by one. We can therefore also compose the representation of subtrees by composing the representation of the head and dependent each time we do an attach-

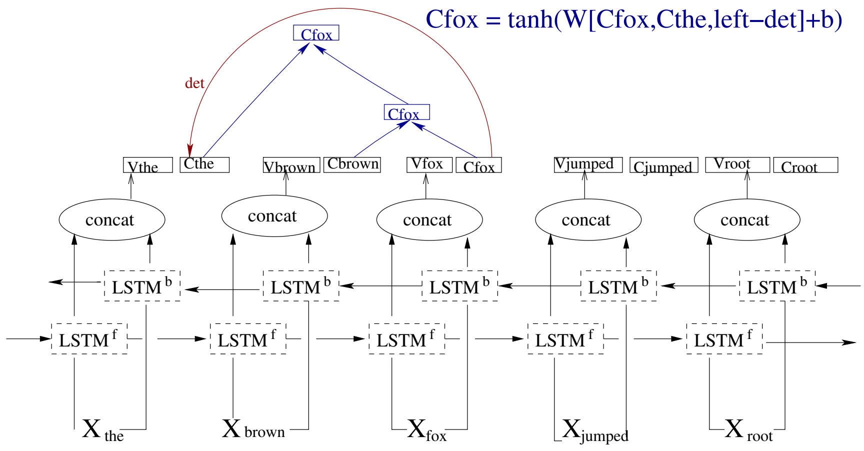


Figure 4.5. TreeRNN parsing architecture.

ment. The representation of words used for making parsing decisions in our parser is their token representation v_i, as defined in Chapter 3. Note that this is different to the S-LSTM which uses a composition function over the *type* representation of words, at least initially. They do not have *token* representations in our sense: a representation of the word which contains information about its context in the sentence. The representation of words in an S-LSTM is initially a vector which contains information about the word type and about its POS tag, as defined in Equation 4.1. This representation gets contextualised as parsing evolves, where word vectors start representing larger subtrees. Note, however, that POS tags do give some information about the word in context: a POS tag is obtained by a POS tagger which has seen the whole sentence. This is a variable that we want to control: we expect composition to be more useful when the model does not have access to POS tags, which give out contextual information about the word.

We create two versions of the parser: one where a word token at index i is represented by a token vector v_i and the other one where it is represented by the token vector and the vector of its subtree at timestep t, c_i^t, as in Equation 4.3, where t is the number of words that have been attached to the subtree. Initially, c_i^0 is just a copy of the token vector (Equation 4.4).

$$v_i = [\text{BiLSTM}(x_{1:n}, i); c_i^t] \tag{4.3}$$

$$c_i^0 = \text{BiLSTM}(x_{1:n}, i) \tag{4.4}$$

When a dependent word d is attached to a word h at index i with a relation and direction r, c_i^t is computed with the same composition function as in the S-LSTM defined in the previous section, repeated in Equation 4.5.[3] The re-

[3]Note that, in preliminary experiments, we tried replacing the vector of the head by the vector of its subtree instead of concatenating the two but concatenating gave much better results.

sulting architecture is depicted in Figure 4.5. The figure shows a configuration resulting from a LEFT-ARC$_{det}$ action, attaching *the* to *fox* and updating the representation c of *fox* by composing it with the representation of *the*. In this configuration, *brown* has already been attached to *fox*, and the representation of *fox* resulting from this configuration represents the subtree *the brown fox*.

This composition function is a simple recurrent cell. Simple RNNs have known shortcomings which have been addressed by using LSTMs, as explained in Chapter 2. A natural extension to this composition function is therefore to replace this recurrent cell with an LSTM cell. We also try this variant. We construct an LSTM for subtrees. We initialise this composition LSTM for each new subtree that is formed, that is, when a dependent d is attached to a head h which does not have any dependent yet. Each time we attach a dependent to a head, we construct a vector which is a concatenation of h, d and r. We pass this vector to the LSTM of h LSTM$_h$. The vector c_h^t is the output state of the LSTM after passing through that vector, as in Equation 4.6. We denote those models with $+rc$ for the one using an ungated recurrent cell and with $+lc$ for the one using an LSTM cell.

$$c_i^t = tanh(W[c_i^{t-1};d;r]+b) \tag{4.5}$$

$$c_i^t = \text{LSTM}_i([c_i^{t-1};d;r]) \tag{4.6}$$

As explained previously, the use of POS tags is a variable we want to control for: we expect composition to be more useful when POS information is not given. As we saw in Chapter 3, POS information and character vectors seem to be partially redundant ways of constructing functional information about the word in context. We therefore also want to test whether or not composition is more useful when character information is not modelled as opposed to when it is. We experiment with the vector representing the word at the input of the BiLSTM. The most complex representation consists of a concatenation of an embedding of the word type $e(w_i)$, an embedding of the (predicted) POS tag of w_i, $e(t_i)$ and a character representation of the word $ce(w_i)$, obtained by running a BiLSTM over the characters of the word. We experiment with ablating either or both of the character and POS vectors. We look at the impact of using composition on the full model as well as these ablated models.

$$x_i = [e(w_i);e(t_i);ce(w_i)]) \tag{4.7}$$

As Section 4.4 will show, composition seems to be mostly superfluous in our parser. For this reason, we investigate the interaction between composition and parts of the network by doing model ablations. We ablate parts of the BiLSTMs: we ablate either the forward or the backward LSTM. We therefore build parsers with 3 different feature functions $f(x,i)$ over the word type vectors x_i in the sentence x (Equation 4.8) to construct the token representation v_i of the word w_i: a backward LSTM (bw) (i.e., ablating the forward LSTM,

Table 4.2. *Hyperparameter values for parsing.*

Word embedding dimension	100
Character embedding dimension	24
Character BiLSTM dimension	50
POS embedding dimension	25
Word BiLSTM Layers	2
Word BiLSTM dimensions	250
Hidden units in MLP	100

Equation 4.9), a forward LSTM (fw) (i.e., ablating the backward LSTM, Equation 4.10) and a BiLSTM (bi) (our baseline, Equation 4.11).

$$v_i = f(x,i) \tag{4.8}$$

$$bw(x,i) = \text{LSTM}(x_{n:1}, i) \tag{4.9}$$

$$fw(x,i) = \text{LSTM}(x_{1:n}, i) \tag{4.10}$$

$$bi(x,i) = [bw(x,i); fw(x,i)] \tag{4.11}$$

K&G parsers with unidirectional LSTMs are, in some sense, more similar to the S-LSTM than those with a BiLSTM, since the S-LSTM only uses unidirectional LSTMs. We hypothesise that composition will help the parser using unidirectional LSTMs in the same way it helps an S-LSTM.

4.4 Experimental Results

4.4.1 Experimental Details

We use our parser described in Chapter 3 with the hyperparameters reported in Table 4.2. We report the average of the 5 best results in 30 epochs and 5 runs with different random seeds. When using POS tags, we use the universal POS tags from the UD treebanks which are coarse-grained and consistent across languages. These POS tags are predicted by UDPipe (Straka et al., 2016) both for training and parsing. Our implementation for this set of experiments is available online.[4]

Data

We test our models on a sample of treebanks from Universal Dependencies v2.1 (Nivre et al., 2017b). We select treebanks according to the criteria described in Chapter 3: we ensure typological variety, we ensure variety of domains, we verify the quality of the treebanks, and we use one treebank with a large amount of non-projective arcs. As argued previously, the inclusion of treebanks with

[4] `http://github.com/mdelhoneux/uuparser-composition`

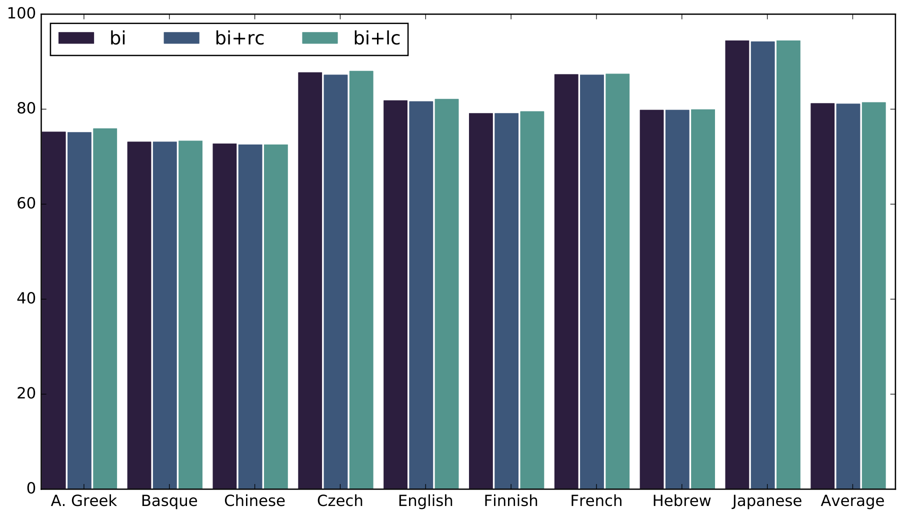


Figure 4.6. LAS of models using a BiLSTM (*bi*) without composition, with a recurrent cell (+*rc*) and with an LSTM cell (+*lc*).

various sizes is more relevant for parser evaluation than development or analysis which is what we are doing here. We therefore remove tiny treebanks from that selection. We add 3 other treebanks instead, adding more typological variety. Our final sample is: Ancient Greek (PROIEL), Basque, Chinese, Czech, English, Finnish, French, Hebrew and Japanese.

4.4.2 Results

First, we look at the effect of our different recursive composition functions on the full model (i.e., the model using a BiLSTM for feature extraction as well as both character and POS tag information). As can be seen from Figure 4.6, recursive composition using an LSTM cell (+*lc*) is generally better than recursive composition with a recurrent cell (+*rc*), but neither technique reliably improves the accuracy of a BiLSTM parser: it improves accuracy for some languages, but decreases it for others, and the difference is small for all languages. This gives negative evidence for **RQ2a**: *Can a BiLSTM parser benefit from recursive subtree composition?* Of course it is possible that we just have not found the way to make composition work with our parsing architecture. We turn to an analysis of the interaction between composition and different parts of the network (**RQ2b**: *How does composition interact with different parts of the network?*) We do this to shed light on the conditions under which composition is useful, since it was shown to be useful in previous work.

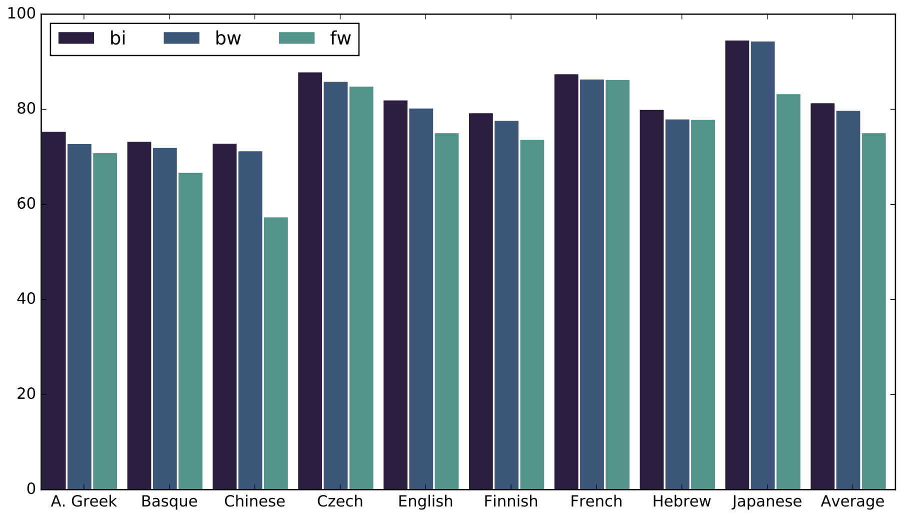


Figure 4.7. LAS of models using a BiLSTM (*bi*), backward LSTM (*bw*) and forward LSTM (fw).

Ablating the Forward and Backward LSTMs

First, we only consider the models using character and POS information and look at the effect of ablating parts of the BiLSTM on the different languages, see Figure 4.7. As expected, the BiLSTM parser performs considerably better than both unidirectional LSTM parsers, and the backward LSTM is considerably better than the forward LSTM, on average. It is, however, interesting to note that using a forward LSTM is much more harmful for some languages than others: it is especially harmful for Chinese and Japanese. This can be explained by language properties: the right-headed languages suffer more from ablating the backward LSTM than other languages. We verify this by investigating the relationship between right-headedness and harmfulness of ablating a backward LSTM in Figure 4.8. We only consider content dependency relations. This is because the UD scheme focuses on dependency relations between content words and treats function words as features of content words to maximise parallelism across languages (de Marneffe et al., 2014), as explained in Chapter 2. We observe a correlation between the difference to the baseline of a forward model and the percentage of right-headed content dependency relations ($R = -0.838$, $p < .01$)

There is no significant correlation between how harmful ablating the forward LSTM is and the percentage of left-headed content dependency relations ($p > .05$), indicating that its usefulness is not dependent on left-headedness. We hypothesise that dependency length or sentence length can play a role but we also find no correlation between how harmful it is to remove the forward LSTM and average dependency or sentence length in treebanks. It is finally

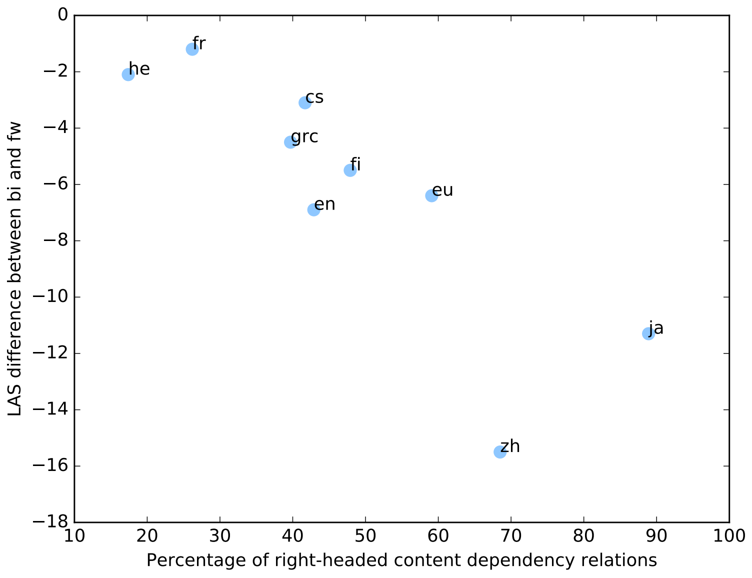


Figure 4.8. Correlation between how harmful it is to remove the backward LSTM and right-headedness of languages.

also interesting to note that the backward LSTM performance is close to the BiLSTM performance for some languages (Japanese and French).

We now look at the effect of using composition with these ablated models. Results are given in Figure 4.9. First of all, we observe unsurprisingly that composition using an LSTM cell leads to higher accuracy than using a simple recurrent cell. Second, both types of composition help the *bw* model, but neither reliably helps the *bi* models: finally, the recurrent cell does not help the fw model but the LSTM cell does to some extent. It is interesting to note that using composition, especially using an LSTM cell, bridges a substantial part of the gap between the *bw* and the *bi* models.

These results can be related to the literature on pre-neural transition-based dependency parsing. Pre-neural transition-based parsers as described in Section 2.3.1 generally rely on two types of features: *history-based features* over the emerging dependency tree and *lookahead features* over the buffer of remaining input. The former are based on a hierarchical structure, the latter are purely sequential. McDonald and Nivre (2007) and McDonald and Nivre (2011) have shown that history-based features enhance transition-based parsers as long as they do not suffer from error propagation. However, Nivre (2006) has also shown that history-based features help but that lookahead features are crucial given the greedy left-to-right parsing strategy.

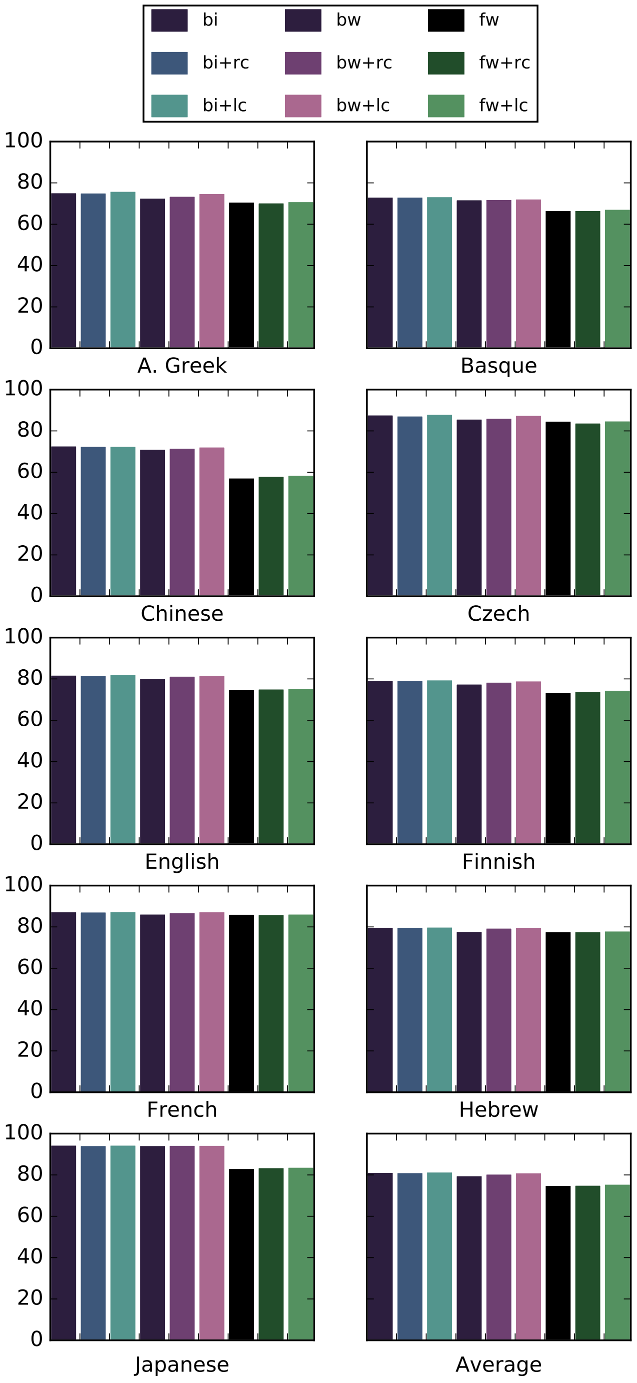


Figure 4.9. LAS when using a BiLSTM (*bi*), backward LSTM (*bw*) and forward LSTM (*fw*), without composition, with a recurrent cell (+*rc*) and with an LSTM cell (+*lc*).

In the model architectures considered here, the backward LSTM provides an improved lookahead compared to pre-neural parsing. Similarly to the lookahead in pre-neural parsing, it is sequential. The difference is that it gives information about upcoming words arbitrarily far away. The forward LSTM in this model architecture provides history-based information but unlike in pre-neural parsing, that information is built sequentially rather than hierarchically: the forward LSTM passes through the sentence in linear order. We see here that lookahead features are more important than the history-based ones. It hurts parsing accuracy more to ablate the backward LSTM than to ablate the forward one. This is expected given that some history-based information is still available through the top tokens on the stack, while the lookahead information is almost lost completely without the backward LSTM.

A composition function gives hierarchical information about the history of parsing actions. It makes sense that this helps the backward LSTM model most since that model has no access to any information about parsing history. It helps the forward LSTM model slightly which indicates that there can be gains from using structured information about parsing history rather than sequential information. We could then expect that composition should help the BiLSTM model which, however, is not the case. This might be because the BiLSTM constructs information about parsing history and lookahead into a single representation. In any case, this indicates that BiLSTMs are powerful feature extractors which seem to capture useful information about subtrees. They seem to make the use of an explicit hierarchical bias superfluous.

Ablating POS and Character Information

Next, we repeat the results from Chapter 3 where we look at the effect of the different word representation methods for the different languages, see Figure 4.10, repeated from that chapter. In all figures and tables, **pos**+ and **char**+ means that the word representations uses a POS embedding and a character representation respectively, **pos**− and **char**− mean that they are not used. As is consistent with the literature, and as was presented in Chapter 3, using character-based word representations and/or POS tags consistently improves parsing accuracy but has a different impact in different languages, and the benefits of both methods are not cumulative. In particular, character models are an efficient way to obtain large improvements in morphologically complex languages.

We investigate the impact of composition on all combinations of ablated models, see Table 4.3. We only look at the impact of using an LSTM cell rather than a recurrent cell since it was a better technique across the board (see Figure 4.9).

Looking first at BiLSTMs (the two first columns of each of the four subtables), composition does not reliably help parsing accuracy, regardless of access to POS and character information: it leads to a maximum of 0.2 LAS points difference on average between bi and $bi+lc$ and a difference above 0.5 in only two cases in all settings (**pos**+**char**+ and **pos**−**char**−, each time for Ancient Greek. This indicates that the vectors obtained from the BiLSTM already con-

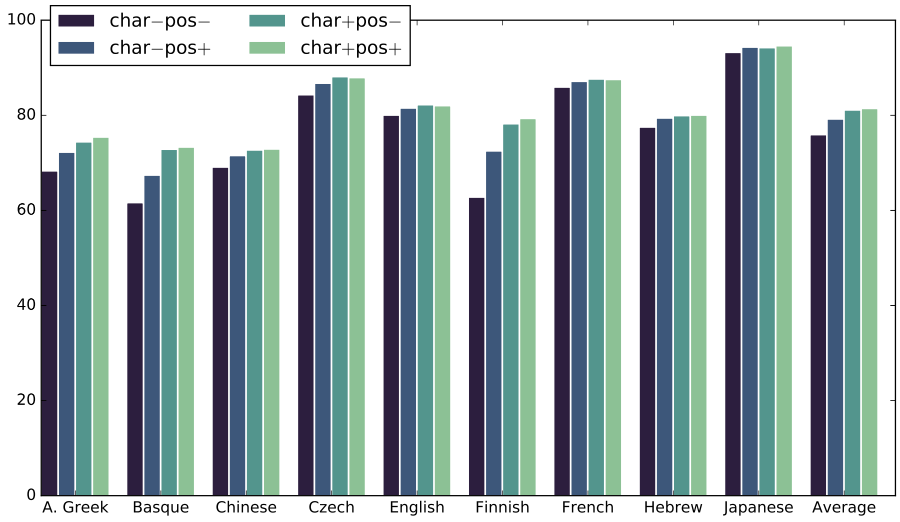


Figure 4.10. LAS of baseline, using char and/or POS tags to construct word representations.

tain information that would otherwise be obtained by using composition.

Turning to results with either the forward or the backward LSTM ablated, we see the expected pattern. Composition helps more when the model lacks POS tags: in the **pos**−**char**+ setting, the model with composition is better by more than 0.5 LAS points for 7 and 6 out of the 9 languages for the *bw* and fw models respectively. In the **pos**−**char**− setting, the model with composition is better by more than 0.5 LAS points for 8 and 6 out of the 9 languages for the *bw* and fw models respectively. This indicates that there is some redundancy between these two methods of building contextual information.

The pattern is similar but not as clear when it comes to character information: the models that lack only character information are not helped as substantially as the models that lack only POS tags information, especially in the case of the fw model where only 5 out of the 9 languages improve by more than 0.5 LAS points. Composition helps the *bw* case in 8 out of the 9 languages but with an average improvement of 1.3 LAS points, which is less than in the case where we only lack POS tags where the average improvement is 1.6 LAS points. This indicates that there is less redundancy between character information and composition than there is between POS tags and composition.

We look more closely at how much of the gap is recovered, on average, by using composition after ablating part of the BiLSTM in Table 4.4 which gives the absolute LAS difference between the *bi* and the *bw* or fw models as well as the percentage of gap recovery from using composition. The first columns for *bw* and fw are the gap between the average performance of the *bi* model and the *bw* or fw models without composition and the second columns are the

Table 4.3. *LAS for bi, bw and fw, without and with composition (+lc) with an LSTM. Difference > 0.5 with +lc in bold.*

	pos+char+						**pos+char−**					
	bi	bi+lc	bw	bw+lc	fw	fw+lc	bi	bi+lc	bw	bw+lc	fw	fw+lc
cs	87.9	88.2	85.9	**87.7**	84.9	85.0	86.7	87.0	84.5	**86.2**	83.6	83.6
en	82.0	82.3	80.3	**81.9**	75.1	**75.6**	81.5	81.5	79.7	**81.4**	74.3	**75.0**
eu	73.3	73.5	72.0	72.4	66.8	**67.4**	67.4	67.6	65.6	**66.3**	59.6	**60.5**
fi	79.3	79.7	77.7	**79.2**	73.7	**74.7**	72.5	72.7	69.8	**71.7**	66.7	**67.4**
fr	87.5	87.6	86.4	**87.5**	86.3	86.4	87.1	87.2	85.8	**86.9**	85.7	85.9
grc	75.4	**76.1**	72.8	**75.0**	70.9	71.1	72.2	72.5	69.6	**71.4**	67.4	67.8
he	80.0	80.1	78.0	**80.0**	77.9	78.2	79.4	79.2	77.2	**79.0**	76.9	77.3
ja	94.6	94.6	94.4	94.5	83.3	**83.9**	94.3	94.3	94.2	94.3	83.0	**83.6**
zh	72.9	72.7	71.3	**72.4**	57.4	**58.7**	71.5	71.3	69.9	**70.8**	56.4	**57.9**
av	81.4	81.6	79.8	**81.2**	75.1	**75.7**	79.2	79.2	77.4	**78.7**	72.6	**73.2**

	pos−char+						**pos−char−**					
	bi	bi+lc	bw	bw+lc	fw	fw+lc	bi	bi+lc	bw	bw+lc	fw	fw+lc
cs	88.1	88.4	86.0	**87.8**	84.7	84.9	84.3	84.5	81.3	**83.1**	79.9	79.8
en	82.2	82.1	79.8	**81.6**	73.2	**73.8**	80.0	79.9	77.5	**79.2**	70.5	**71.5**
eu	72.8	72.9	71.5	71.8	65.4	**66.4**	61.6	62.0	57.7	**59.5**	48.7	**51.2**
fi	78.2	78.6	75.8	**77.9**	72.0	**73.0**	62.8	63.1	56.6	**60.2**	52.8	**54.7**
fr	87.6	87.7	86.1	**87.4**	85.4	85.7	85.9	85.8	83.7	**85.3**	83.1	83.3
grc	74.4	74.8	71.3	**73.7**	69.2	69.6	68.3	**69.0**	64.6	**67.3**	62.6	**63.4**
he	79.9	80.1	77.4	**79.9**	76.5	**77.3**	77.5	77.4	74.4	**77.2**	74.2	74.7
ja	94.2	94.4	94.2	94.4	81.3	**81.8**	93.2	93.3	92.7	93.1	79.5	**80.2**
zh	72.7	72.5	70.8	**72.2**	56.5	**58.2**	69.1	69.3	66.7	**68.1**	53.4	**55.0**
av	81.1	81.3	79.2	**80.8**	73.8	**74.5**	75.9	76.0	72.8	**74.8**	67.2	**68.2**

gap between the average performance of the *bw* or fw models with and without composition. First, we can see that the gap recovery from using composition with the fw model is very small compared to with the *bw* model. Second, we can see that in terms of absolute differences, composition helps most when there is no access to POS tags (the last two rows) but in terms of gap recovery, it helps most when characters are present in the model, indicating further that character vectors and composition are less redundant than other types of information. It means that these two types of information are still complementary and a recursive composition function cannot fully compensate for the lack of character information.

There are some language idiosyncrasies in the results. While composition helps recover most of the gap for the backward LSTM models without POS and/or character information for Czech and English, it does it to a much smaller extent for Basque and Finnish. We hypothesise that arc depth might impact the usefulness of composition, since more depth means more matrix multiplications with the composition function. However, we find no correlation between average arc depth of the treebanks and usefulness of composition. It is an open question why composition helps some languages more than others.

Table 4.4. *Gap between bi and bw or fw and gap recovery from using composition (+lc) in absolute LAS points and in % (%rec.).*

	bi-bw	[bw+lc]-bw	%rec.	bi-fw	[fw+lc]-fw	%rec.
pos+char+	1.6	1.4	87.5	6.3	0.6	9.5
pos+char−	1.8	1.3	72.2	6.6	0.6	9.1
pos−char+	1.9	1.6	84.2	7.3	0.7	9.6
pos−char−	3.1	2	64.5	8.7	1	11.5

Overall, we have observed that recursive composition is partially redundant with the forward part of the BiLSTM, as well as with POS tag information, providing answers to **RQ2b**: *How does recursive composition interact with different parts of the network?*

Recurrent and recursive LSTMs in the way they are considered here are two ways of constructing contextual information and making it available for local decisions in a greedy parser. The strength of recursive LSTMs is that they can build this contextual information using hierarchical context rather than linear context. A possible weakness is that this makes the model sensitive to error propagation: an incorrect attachment leads to using the wrong contextual information. It is therefore possible that the benefits and drawbacks of using this method cancel each other out in the context of BiLSTMs.

In general, it seems that using BiLSTM vectors of a few elements of the stack and buffer is a very strong baseline which might mean that these vectors contain all the necessary information for scoring transitions. There is more evidence of this: Falenska and Kuhn (2019) found that structural features such as children of head vectors are superfluous in this parsing architecture: they do not improve parsing accuracy. They showed this for both the transition-based and the graph-based version of the parser from Kiperwasser and Goldberg (2016b). Similarly, Gontrum (2019) found that an attention layer over the elements of the stack and the buffer does not improve transition-based parsing, indicating again that the information needed for scoring transitions is present in the BiLSTM representations of top elements of the stack and the first element of the buffer. Hewitt and Manning (2019) investigated contextual word representations learned by BERT (Devlin et al., 2019), a language model trained on English using a large dataset, and found that tree structures can be read off directly from these representations. As far as I am aware, no one has investigated whether contextual representations obtained from a BiLSTM trained for the task of parsing encode entire subtrees but it is conceivable that they do.

Ensemble

To investigate further the information captured by BiLSTMs, we ensemble the 6 versions of the models with POS and character information with the different feature extractors (*bi*, *bw*, fw) with (+lc) and without composition. We use

Table 4.5. *UAS ensemble (full) and ablated experiments.*

	bi	full	-bi	-[bi+lc]	-bw	-[bw+lc]	-fw	-[fw+lc]
cs	90.9	92.0	91.8	91.8	91.8	91.8	92.1	92.0
en	85.8	87.1	86.7	86.7	86.8	86.7	87.2	87.2
eu	78.7	80.9	80.3	80.2	80.4	80.3	80.9	81.0
fi	83.5	85.5	85.4	85.4	85.3	85.2	85.6	85.5
fr	89.8	90.8	90.8	90.6	90.8	90.7	90.8	90.8
grc	81.2	83.5	83.0	83.1	83.3	83.0	83.4	83.6
he	86.2	87.6	87.6	87.4	87.5	87.2	87.6	87.7
ja	95.9	96.1	95.8	95.7	95.9	95.8	96.3	96.2
zh	78.3	79.3	78.4	78.6	78.4	78.7	79.8	79.9
av	85.6	87.0	86.6	86.6	86.7	86.6	87.1	87.1

the (unweighted) reparsing technique of Sagae and Lavie (2006)[5] and ignoring labels. As can be seen from the UAS scores in Table 4.5, the ensemble (*full*) largely outperforms the parser using only a BiLSTM, indicating that the information obtained from the different models is complementary. To investigate the contribution of each of the 6 models, we ablate each model one by one. Table 4.5, ablating either of the BiLSTM models or the backward LSTM using composition, results in the least effective of the ablated models, further strengthening the conclusion that BiLSTMs are powerful feature extractors.

4.5 Conclusion

In this chapter, the goal was to find out if we can make our parsing model more linguistically informed by incorporating a hierarchical bias in the parsing architecture such as has been shown to work well in the S-LSTM. This is to provide an answer to **RQ2**:*How can we incorporate linguistic knowledge into neural models for dependency parsing?*

The usefulness of a hierarchical bias was first motivated linguistically and from a typological perspective. I discussed work that has shown that LSTMs are capable of learning hierarchical structure in spite of being sequential models as well as work that has shown that these results might have been tied to a specific type of language or restricted to a specific type of syntactic structure. I mentioned studies that have shown that recursive neural networks are better at capturing syntactic structure than recurrent neural networks. These studies led us to hypothesise that our parsing model should benefit from using a recursive layer, similarly to how it helps the S-LSTM. This hypothesis was not confirmed: our model performs similarly with and without the recursive layer.

[5]This method scores all arcs by the number of parsers predicting them and extracts a maximum spanning tree using the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967).

We did extensive analysis to tease out the conditions under which a tree layer is useful. We found that it is a useful way of constructing information, similar to history-based features in pre-neural dependency parsing, but that a forward LSTM which is part of a BiLSTM seems to be better at this because BiLSTM parsers perform better than a parser with composition.[6] This explains why composition is useful in the S-LSTM which only uses unidirectional LSTMs and not in a K&G parser. I discussed the benefits and drawbacks of a tree layer in the way it is considered here. The advantage is that it uses syntactic recency as opposed to sequential recency, which is likely to be more relevant for making parsing decisions. The drawback is that it is sensitive to error propagation: if we draw an incorrect arc, we will have a poor representation of the word. The advantages and drawbacks seem to cancel each other out in our parser when we look at parsing accuracy. Composition may be beneficial in some circumstances and detrimental in others. In Chapter 5, we consider the case where it is used only to model a specific type of subtree: a dissociated nucleus. We will see the benefits of doing so and hypothesise that we just have not found the best way to incorporate a hierarchical bias in our model yet, but that there might still be benefits from doing so.

[6]It is plausible that the parser does not construct information like history-based features but functions differently. This is an open question which we leave to future research.

5. Auxiliary Verb Constructions

This chapter focuses on auxiliary verb constructions. I start by describing AVCs in dependency grammar, dependency parsing and UD in Section 5.1, repeating some of the notions that were introduced in Chapter 2 that are essential here. In Section 5.2, we look at the impact of using a UD representation of AVCs, where main verbs are the head, as opposed to a representation where auxiliaries are the head. There has been extensive work investigating the impact of different representations of AVCs, among other constructions, on parsing accuracy and it was believed for some time that the UD representation is suboptimal for parsing. However, we have recently shown that it is actually a better representation than the alternative. Previous work, including ours, investigated this question using pre-neural parsers. Here, we investigate what happens with our neural parser. We then investigate what our parsing model should, does and can learn about AVCs in Section 5.3, focusing on the question of whether or not it can learn the notion of dissociated nucleus. We look at the usefulness of a recursive composition function as described in Chapter 4 in learning that notion. This last study is an attempt at answering **RQ3**: *How can we investigate what neural models learn about language when trained for the task of dependency parsing?*

5.1 Auxiliary Verb Constructions in UD

In Chapter 2, I introduced dependency grammar and its use in dependency parsing. As described in that chapter, the different dependency grammar traditions all agree that most relations that hold between words in a sentence are asymmetric dependency relations between a head and a dependent. For practical purposes, dependency parsing has been working with a definition of dependency trees where *all* relations that hold between words in the sentence are asymmetric dependency relations. With this definition, words are the basic unit of syntax. Representations in most dependency grammars are more complex. In Tesnière (1959), the basic unit of syntax is the *nucleus*, which can correspond to a word but can also consist of several words. When consisting of several words, the nucleus is called a *dissociated nucleus* and its internal elements are connected by *transfer* relations. The nuclei themselves are connected by *dependency* relations or (in the case of coordination) by *junction* relations. Transfer relations typically connect a content and a function word.

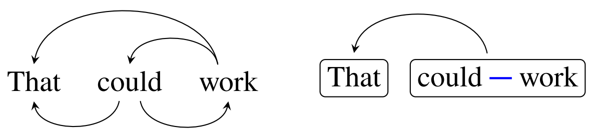


Figure 5.1. Two different representations of a sentence with auxiliary as used in dependency parsing (left) vs as can be represented following Tesnière (1959) (right).

This chapter focuses on auxiliary verb constructions (AVCs), as they are a typical example of a dissociated nucleus and are well attested typologically; see for example Anderson (2011). In AVCs, head properties are shared by one or several auxiliaries and the main verb. Inflectional verbal features like agreement, tense, aspect, mood, etc. are typically encoded in the auxiliary whereas lexical features like valency are properties of the main verb. The relation that holds between them is a relation of *transfer*, which involves changing the grammatical category of a content word (Tesnière, 1959) by for example changing the tense or number of the main verb.

In Chapter 2, I gave an example tree representation for an AVC, repeated in Figure 5.1. In the dependency parsing literature, a sentence with an auxiliary is usually represented as either the top or the bottom tree in the left part of Figure 5.1, with either the auxiliary or the main verb being dependent on the other. If we follow the ideas of Tesnière (1959), it can be represented as in the right part of Figure 5.1 where the auxiliary and main verb are connected by a transfer relation to form a nucleus. This nucleus is itself connected to the other words/nuclei in the sentence. In this example, the word *That* corresponds to a nucleus and the words *could* and *work* are each part of a dissociated nucleus.

As also described in Chapter 2, Universal Dependencies has a fundamental design principle that dependency relations primarily hold between content words. This has the consequence that function words normally attach to content words. For AVCs, the consequence is that the auxiliary attaches to the main verb, as in the top tree of the left part of Figure 5.1. This is a controversial decision for theoretical and practical reasons. It is controversial for theoretical reasons since most dependency grammars treat function words as heads, at least in surface syntax (Osborne and Maxwell, 2015), and would favour a representation where the auxiliary is the head, as in the bottom tree of the left part of Figure 5.1. However, if we follow the ideas of Tesnière (1959)'s, auxiliary verb constructions, among other constructions involving function and content words, form part of a dissociated nucleus and the relation that holds between the main verb and the auxiliary is a relation of transfer, not of dependency. Choosing a head is therefore just a matter of convenience, it allows us to keep this representation as part of a tree, and it does not matter which one we choose. As argued by de Marneffe and Nivre (2019), if we read the label *aux* as indicating a relation of transfer rather than a relation of dependence, we can interpret the tree representation of auxiliary verb constructions as a dissociated nucleus. The choice of attaching content words to function words is also controversial for practical reasons, since some work (Schwartz et al. (2012) among others) has provided evidence that having function words as head of a relation between a content and a function word is better for parsing accuracy in some cases, including AVCs. This evidence is mixed, however, as will be described in Section 5.2 and we found in de Lhoneux and Nivre (2016) that it is best to have main verbs as heads in AVCs.

5.2 AVC Representation for Parsing

As just mentioned, there is some evidence in the literature that it is better to attach content words to function words than the other way around for parsing accuracy. For example, Schwartz et al. (2012) investigate the representations of six different constructions in English. They modify the annotation of each construction and compare parsing accuracy with the two different representations across five parsers. They find substantial differences in accuracy with different representations. Their results show that having function words as heads of a content function dependency relation leads to higher accuracy in some cases, lower in other cases. It is substantially better for parsing accuracy to have the preposition as head of prepositional phrases than the noun and it is slightly better to have auxiliaries as head of verb group constructions than having the main verb as head. By contrast, it is substantially better to have nouns as heads of nominal phrases than determiners and to have conjuncts as heads of coordinating constructions. Similarly, Nilsson et al. (2006) and Nilsson et al. (2007) show that it is beneficial for parsing to transform the representation of AVCs so that the auxiliary is the head in three treebanks where AVCs are annotated with the main verb as head. Their technique is different from the one in Schwartz et al. (2012): Schwartz et al. (2012) have no a priori preference for a representation and their study is aimed at finding out which one is the best. They aim to find out which one is the most *learnable*. By contrast, Nilsson et al. (2007) want to improve accuracy on four treebanks in their original annotation scheme (the PDT scheme). They therefore transform the representation of AVCs in a pre-processing step, train and test a parser with that representation and subsequently attempt to transform back to the original representation in a post-processing step. de Marneffe et al. (2014) made the design choice of having function words attach to content words in spite of the partial evidence that it is suboptimal for parsing but suggesting that the method from Nilsson et al. (2007) can be used to improve parsing accuracy. They suggest that we can have an intermediate representation for the purpose of parsing.

Silveira and Manning (2015) attempted a similar approach with the UD English EWT treebank with different constructions (not including AVCs) with mixed success. They find that while the transformed representation is in some cases easier to learn than UD (they obtain better accuracy when parsing with this representation), transforming that representation back to UD leads to drops in accuracy. They hypothesise that this is due to error propagation when performing backtransformation.

The study by Silveira and Manning (2015) together with other recent studies show mixed results which cast doubt on the hypothesis that UD representations are generally bad for parsing. Rosa (2015) shows that the PDT representation of prepositional phrases is better than the UD representation when training monolingual parsers, but the UD representation is better when doing parser transfer, indicating that it is a better cross-lingual abstraction. Attardi et al. (2015) show that the UD representation of prepositional phrases and copulas is better than alternative representations for Italian. Wisniewski and Lacroix (2017) show that the UD representation is better than an alternative for many constructions excluding AVCs (subordinate clauses, noun phrases, noun sequences, prepositional phrases, coordination and copulas) for many UD treebanks.

In de Lhoneux and Nivre (2016), we found that the UD representation of AVCs is better than the alternative for parsing accuracy in different respects: using it as an intermediate representation as was done in Nilsson et al. (2006) and Nilsson et al. (2007) does not lead to improved parsing accuracy, and it is more learnable than the alternative in the sense of Schwartz et al. (2012). We found evidence that the reason this approach worked for Nilsson et al. (2006) and Nilsson et al. (2007) is that, in their case, it helps to disambiguate POS tags: it makes it clear what verbs are auxiliaries and what verbs are main verbs. All these studies have been done using pre-neural parsers. In the next section, we verify the finding that UD is a better representation than the alternative for parsing accuracy with our neural architecture. Given that the BiLSTM gives the parser better access to global structure than pre-neural parsers, and given that they made the use of POS tags almost superfluous, we hypothesise that the different representations will have a much smaller impact on parsing accuracy than it did in previous studies. For this, we repeat our experiments from de Lhoneux and Nivre (2016) using our parser. I explain the methodology from that study in detail first.

5.2.1 AVC Representations Comparison

Nilsson et al. (2006), Nilsson et al. (2007) and de Lhoneux and Nivre (2016) compared two representation style of AVCs (among other constructions for the first two studies): the representation from the Prague Dependency Treebank (henceforth PDT) which is the same as the UD representation, and a representation described in Mel’čuk (1988) (Mel’čuk style, henceforth MS). These are represented in Figure 5.2 (they correspond to the top and bottom tree of the left part of Figure 5.1 respectively.) We can deterministically transform one rep-

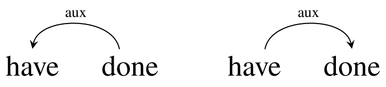


Figure 5.2. PDT/UD (left) and MS (right) representation of an AVC.

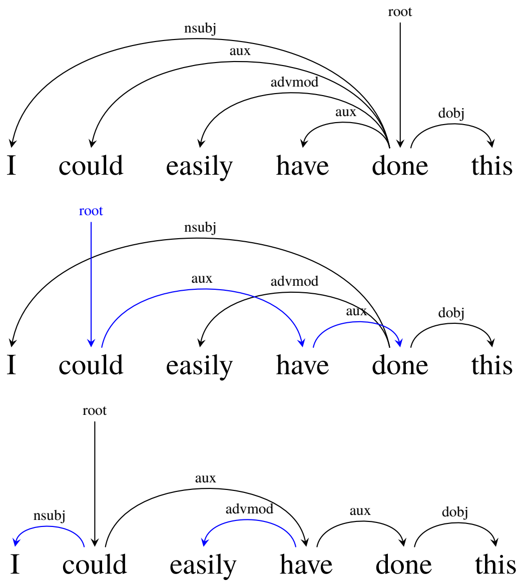


Figure 5.3. Example 5 annotated in UD (top), in an intermediate representation (middle) and in MS (bottom). Thick blue dependencies in a tree are these that changed compared to the tree above it.

resentation to the other. I describe the transformation (PDT/UD to MS) and backtransformation (MS to PDT/UD) algorithms in turn.

Transformation Algorithm

The transformation algorithm is illustrated by Figure 5.3, which represents the transformation of a sentence with an AVC given in Example (5). The top tree is the original UD representation of this example, the middle tree an intermediate representation and the bottom tree is the final MS representation.

(5) I could easily have done this

The transformation first looks for AVCs in a dependency graph. Those AVCs are collected in the set V. An AVC v_i has a main verb mv_i (*done* in the example) and a set of auxiliaries AUX_i with at least one element (*could* and *have* in the example). AVCs are collected by traversing the sentence from left to right, looking at auxiliary dependency relations. An auxiliary dependency relation $w_{aux} \xleftarrow{aux} w_{mv}$ is a relation where the main verb is the head and the auxiliary is the dependent. Only auxiliary dependency relations between two verbal forms are considered.[1] When such a dependency relation is found, if there is a v_i in V that has the head of the dependency relation (w_{mv}) as main verb mv_i, w_{aux} is added to that v_i's set of auxiliaries AUX_i. Otherwise, a new v, v_{i+1}, is created and added to V.

After that, for each v_i in V, if there is only one auxiliary in AUX_i, the dependency relation between that auxiliary and the main verb mv_i is inverted and the head of mv_i becomes the head of the auxiliary. When there are several auxiliaries (like in example (5)), the algorithm attaches mv_i to the closest one and the head of mv_i becomes the head of the outermost one.[2] Any auxiliary in-between is attached in a chain from the outermost to the one that is closest to the verb. In the example, the main verb *done* gets attached to the closest auxiliary *have* and the head of the main verb *done* which was the root becomes the head of the outermost auxiliary, *could*.

Next, we reattach dependents of the main verb, making sure that projectivity is maintained. As a matter of fact, as can be seen from the middle tree in Figure 5.3, the previous changes can introduce non-projectivity into an otherwise projective tree, which is undesirable, for reasons explained in Chapter 3. Dependents to the left of the leftmost verb of the whole AVC (i.e. including the auxiliaries and the main verb) get attached to the leftmost verb. In the example, *I* gets attached to *could*. Dependents to the right of the rightmost verb of the AVC get attached to the rightmost verb. In the example, *this* remains attached to the main verb *done*. Any remaining dependent gets attached to the auxiliary that is closest to the main verb.[3] In the example, *easily* gets attached to *have*. Another possibility which would make the two representations more of a mirror image than they are here would be to attach all dependents to the outermost auxiliary. However, the method just described is closest to what is done in Nilsson et al. (2006) and we do not want to add unnecessary confounding factors to our experiments. Nilsson et al. (2006) do not specify what happens to middle dependents but for the rest, they do the same as just described.

Backtransformation Algorithm

The backtransformation algorithm works similarly to the transformation algorithm. A set of AVCs V is first collected by traversing the sentence from left to right, looking at auxiliary dependency relations. An auxiliary dependency relation $w_d \xleftarrow{aux} w_h$ between a dependent w_d and a head w_h in MS can be between an auxiliary and the main verb or between two auxiliaries. When one such relation is found, if its head w_h is not already in a aux_i in V, a new AVC

[1]Nouns can be the heads of auxiliary dependency relations in copula constrictions in UD and we want to filter out those cases to make sure we have a main verb.

[2]In case of a tie (if there are auxiliaries on both sides of the verb with equal distance to it), we define the leftmost auxiliary as the outermost one.

[3]In case of a tie between an auxiliary to the left and one to the right of the main verb, the dependent gets attached to the left one.

V_i is created and w_h is added to aux_i. When we find an auxiliary dependency relation, if its dependent is not in the list of auxiliaries already processed, we follow the chain of heads until we find an auxiliary which is not itself the dependent of an auxiliary relation. We then follow the chain of dependents until we find a node which is not the head of an auxiliary dependency relation. That node is the main verb. While recursing the auxiliary chain, we add each head of an auxiliary dependency relation to the list of auxiliaries for the sentence. After collecting the AVCs of the sentence, re-attaching the nodes is straightforward. The main verb gets re-attached to the head of the outermost auxiliary and all auxiliaries and their dependents get re-attached to the main verb. In the previous example, the bottom tree of Figure 5.3 can be transformed back to the top tree in this way: *done* is identified as the main verb of the AVC and *could* as its outermost auxiliary. The head of *could* therefore becomes the head of *done* and the two auxiliaries of the sentence as well as their dependents get attached to the main verb. Note that the algorithm assumes a well-formed tree and has no way of recovering from parser errors. For example, if, during parsing, an auxiliary dependency relation is constructed with a noun as dependent, the noun will incorrectly be assumed to be a main verb by the algorithm.

Representation Comparison

Equipped with these transformation and backtransformation algorithms, we can compare a parser trained on the original UD representation with a parser trained on the MS transformed representation. It is also informative to do further comparisons to better understand the factors at play. In Figure 5.4, I summarise the four different pipelines that we tested in de Lhoneux and Nivre (2016), which I name ud2ud ms2ud, ms2ms and ud2ms. We further use these pipelines here. In ud2ud, we train the model and parse the test set in the original UD representation. In ms2ud, we transform the training data from UD to MS, train, parse, and transform the parsed data back to UD. Comparing ud2ud and ms2ud is the main comparison we are interested in: we want to know how well we can do with a UD representation. In ms2ms, we transform the training and test data from UD to MS, train and parse in the MS representation and then evaluate on the MS representation. In ud2ms, we train on the UD representation, parse and transform the parsed data from UD to MS and evaluate on the MS representation.

As mentioned previously, Schwartz et al. (2012) do not transform the representation back to the original style; they instead compare parsing accuracy of the models trained on the two different representations directly on test sets that are in the respective style. This is with a different purpose in mind: they want to find out which representation is more *learnable*. The question of learnability is also interesting here, it can shed further lights on the representation comparison. Testing learnability involves comparing ud2ud and ms2ms.

As mentioned earlier, Silveira and Manning (2015) is a study similar to what we do here: they transform the representation of several constructions from

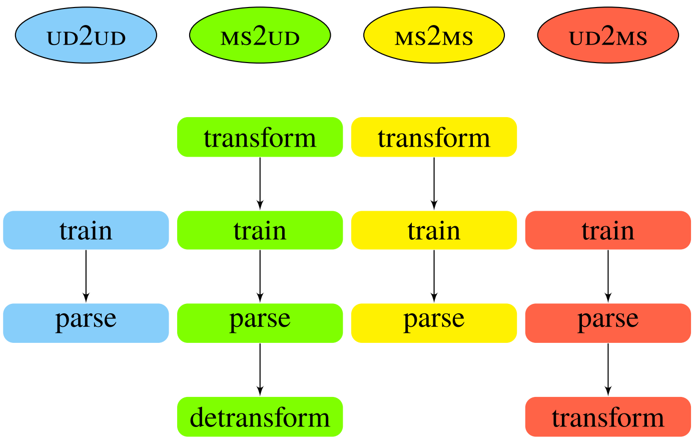


Figure 5.4. Pipelines.

UD to make function words the head of content function relations. They find that while parsers trained with some of the transformed representations perform better on their own gold standard than parsers trained with the original representation on the original gold standard, transforming it back to the original representation leads to drops in accuracy. They conclude that the problem comes from the backtransformation approach. They hypothesise that a problem with backtransformation is that it can lead to what they call *error amplification*, where one error leads to many more. A transformation can be prompted by the presence of only one dependency relation but involve transformations of many surrounding dependency relations. This can be seen in Figure 5.3 where the transformation re-attaches most of the words in the tree. If, then, an incorrect dependency relation prompts a transformation in the parsed data, its surrounding items which might have been correct become incorrect. An incorrect parse can then become worse. We can verify whether or not error amplification is the main explanation for the drop in accuracy when using the transformed model by comparing the difference in accuracy between the model trained on the transformed and the original representation on the original data (UD2UD and MS2UD) and the difference in accuracy between the model trained on the original and transformed representation on the transformed representation (MS2MS and UD2MS). Since, as reported in the next section, our backtransformation method is almost perfectly accurate on gold trees with a correct annotation, the backtransformation algorithm itself does not introduce errors. If error amplification is the main explanation, we would therefore expect to observe symmetry in these differences: backtransformation will hurt the transformation from MS to UD as much as it hurts the transformation from UD to MS.

Nilsson et al. (2006) and Nilsson et al. (2007) found MS to be a better parsing representation than PDT/UD: UD2UD is better than MS2UD. We found the

Table 5.1. *Hyperparameter values for parsing.*

Character embedding dimension	24
Character BiLSTM output dimension	50
Word embedding dimension	100
Word BiLSTM layers	2
Word BiLSTM hidden/output dimension	250
Hidden units in MLP	100

opposite to be true in de Lhoneux and Nivre (2016). We found some tentative explanations of why these previous studies found that MS2UD is better than UD2UD: the MS representation seems to help the parser in disambiguating POS tags, which helps in identifying main verbs and auxiliaries. We further found that UD is more learnable than MS and that error amplification does not seem to be the main issue with the MS2UD model. Our findings in de Lhoneux and Nivre (2016) can be summarised as follows:

- UD is a better representation than MS for parsing (UD2UD is better than MS2UD)
- UD is a representation which is easier to learn than MS (UD2UD is better than MS2MS)
- Error amplification is not the main problem with having a transformed parsing representation (the differences between UD2UD and MS2UD is not similar to the difference between MS2MS and UD2MS.)

The questions we ask in this section are as follows (they are numbered with roman numerals because they are not central to the thesis and not directly related to our general research questions):

RQI Does the representation of AVCs matter in neural parsing?
RQII Is UD a better representation than MS for parsing accuracy with a neural parser?

5.2.2 Experiments

We train parsers for 30 epochs with the hyperparameters reported in Table 5.1 and report the score of the best epoch on the development set.

Data

We follow our usual sampling criteria, with additional criteria. We construct a sample for the two studies presented in this chapter, with some criteria that are only relevant for the second study. The selection procedure is therefore discussed in Section 5.3. We select Catalan (AnCora), Croatian (SET), Dutch (Alpino) and Finnish (TDT).

Table 5.2. *LAS with the 4 different pipelines and difference in accuracy between pairs. Bold indicates the best between the first and second column. Italics indicates the best between the first and third column. Significance for the McNemar test is indicated with* $* = p < .05$ *and* $** = p < .01$.

Language	UD2UD	MS2UD	δ	MS2MS	UD2MS	δ
Catalan	***87.8***	87.4**	-0.4	*87.8*	87.9	0.1
Croatian	***81***	80.4*	-0.6	80.5	80.8	0.4
Dutch	***84.1***	84	-0.1	*84.1*	81.2	-2.9
Finnish	***78.9***	78.5*	-0.4	78.6	77.4	-1.2
Average	82.9	82.6	-0.4	82.7	81.8	-0.9

Results

Results of all experiments are given in Table 5.2. Significance levels are reported for the difference between the two first columns, UD2UD and MS2UD. These are significance levels for the McNemar test as obtained by MaltEval (Nilsson and Nivre, 2008). The corresponding results from de Lhoneux and Nivre (2016) using MaltParser and all treebanks of UD v1.2 (Nivre et al., 2015) are in Appendix C. We report the macro-average of results for the four languages. We test the accuracy of our backtransformation algorithm by doing a round trip transformation and backtransformation experiment on the training sets. We obtain 99.4%, 100%, 100% and 99.9% LAS for Catalan, Finnish, Croatian and Dutch respectively. We had obtained accuracies of 100% for most of treebanks in v1.2. This indicates that the transformation and backtransformation algorithms are fully deterministic given a correctly annotated tree.

We can see that the difference between the UD2UD and the MS2UD models is smaller than in de Lhoneux and Nivre (2016). In that paper, UD2UD was better than MS2UD on average by 0.8 LAS points (Appendix C), here it is only 0.4 LAS points. However, this difference is still significant for 3/4 languages, indicating that the representation of AVCs does still matter in neural parsing. This answers **RQI** positively: *Does the representation of AVCs matter in neural parsing?*

We can also see that our finding from de Lhoneux and Nivre (2016) still holds: UD is a generally better representation for parsing accuracy than MS: UD2UD is better than MS2UD. This answers **RQII** positively: *Is UD a better representation than MS for parsing accuracy with a neural parser?*

In de Lhoneux and Nivre (2016), the average difference between UD2UD and MS2MS was 0.7, here it is only 0.2, indicating that the two representations are almost equally learnable for a neural parser. In de Lhoneux and Nivre (2016), UD2MS was better than MS2MS by 0.7 LAS points on average and this surprising finding does not hold here where we see a general decrease from using the backtransformation approach, if we look at average accuracies. However, there is still no symmetry between the differences from using or not using backtransformation and the picture is more mixed here than it was in de Lhoneux and

Nivre (2016): the accuracy decreases substantially from ms2ms to ud2ms and increases for the other two but by a small margin. It would be interesting to look at more languages to get a clearer picture of what is happening.

5.3 What Does a Neural Parser Learn about AVCs?

As described in Section 5.1, in the work of Tesnière (1959), AVCs are an example of a dissociated nucleus and the relation between the main verb and the auxiliary is a relation of transfer. As also described previously, UD chose a representation which is compatible with this interpretation. However, this has not been made explicit when training parsers. In pre-neural transition-based parsers like Nivre et al. (2006), when a dependent gets attached to its head, features of the head are still used for further parsing but features of the dependent are usually discarded.[4] In neural parsers, it is less clear what information is used by parsers. As described at length in this thesis, current state-of-the-art models use (Bi)LSTMS, and LSTMs make it possible to encode information about the surrounding context of words in an unbounded window (which is usually limited to a sentence in practice). As also described in Chapter 2, various methods have been developed to interpret what neural models learn, including the use of diagnostic classifiers, which is what we use here. In particular, we are interested in finding out whether or not our parsing models learn the notion of dissociated nucleus. We can use diagnostic classifiers to look at whether information like valency, agreement, tense, mood, etc. is encoded in a vector representing a word or a subtree in parsing. This allows us to make the main research question more specific: when making a parsing decision about a nucleus, does the parser have access to similar information regardless of whether the nucleus is dissociated or not? For the case of AVCs, this means that the parser should learn similar information about AVCs as it does about their non-dissociated counterpart: simple finite main verbs (henceforth FMV).

We saw in the previous section that the UD representation which was believed to be suboptimal for parsing is in fact better than one alternative where auxiliaries are the head of AVCs. We are therefore now primarily interested in using this representation. However, we still find it interesting to investigate the impact of using a different representation on what the parser learns about AVCs, and also experiment with an MS representation of AVCs.

5.3.1 Prediction Tasks

As described in Chapter 2, the method of diagnostic classifiers is used to test whether or not some information is in a set of vectors. To do this, a task is defined and the set of vectors is trained for that task. If we can successfully train

[4]Although features of the dependent can be used as features of the head.

a classifier on the set of vectors for the task, we have some indication that the vectors contain this information. For example, Adi et al. (2017) test sentence vectors on several tasks, including testing whether or not a sentence vector encodes its sentence length.

With UD treebanks, it is straightforward to design tasks that probe transitivity in AVCs and FMVs: we can look at whether the main verb has objects and indirect objects. It is also straightforward to design tasks that probe agreement of AVCs and FMVs: we can use the morphological features that encode information about the subject's number and person. It is less straightforward to design tasks that probe information about tense, mood, and aspect (TMA) because that would require annotation of the verb phrases, since the morphological features of individual verbs do not give enough information: they give information at the lexical level. For example, in the AVC *has been*, the tense feature for *has* is *present* and for *been* it is *past* but no feature indicates that the AVC is in the present perfect.[5] We therefore leave TMA features to future work and use agreement and transitivity tasks and look at whether or not subtrees representing AVCs encode the same information about these tasks as FMVs. Note that there is a difference between transitivity as a lexical property of a verb and transitivity of a specific clause: some verbs can be used both transitively and intransitively (Dixon and Aikhenvald, 2000). For practical reasons, we only consider transitivity as a property of a clause, rather than as a lexical property of a verb.

An assumption underlying our question is that agreement and transitivity information is learned by the parser and specifically, that it is available to the parser when making decisions about main verbs. Johnson et al. (2011) found that adding subject-agreement features in the Charniak parser did not improve accuracy but explained this result by the fact that this information was already present in POS tags. We do not use POS tags in our parser and it seems reasonable to assume that agreement information is useful for parsing, which we, however, want to verify.

We compare how well the FMV representations do on these tasks to a reasonable baseline: the majority baseline (the most frequent value for the task in the training part of a treebank). We also want to compare the representation of FMVs with the representation of a nearby token that is not expected to have this information, to rule out the possibility that the LSTM propagates information about AVCs to all the sentence tokens or at least the nearby ones. We select punctuation marks that are close to the main verb for this purpose. Punctuation items attach to the main verb in UD and are frequent enough that we can expect many main verbs to have at least one as a dependent. In addition, unlike other items of the verb phrase, vectors representing punctuation marks are not expected to have information about either of the two tasks. We also compare the verb token vectors (the vectors obtained by the BiLSTM which

[5] See https://universaldependencies.org/u/feat/Tense.html

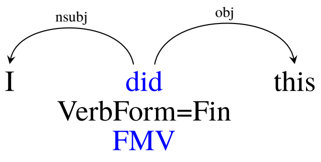


Figure 5.5. Finite main verb in a UD tree.

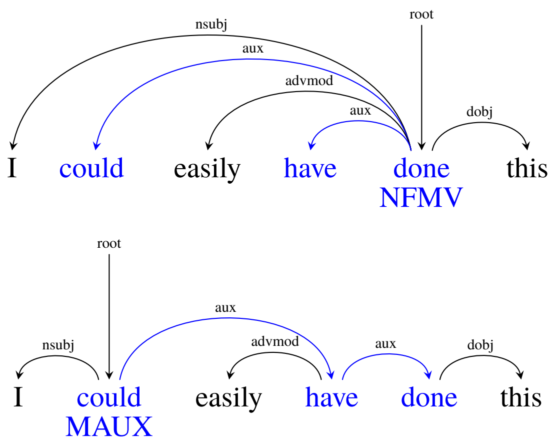


Figure 5.6. Example sentence with an AVC annotated in UD (top), and in MS (bottom). AVC subtree in thick blue.

therefore represent the word in the context of the sentence) to the corresponding vectors for verb types (the vectors at the input of the BiLSTM) in order to better understand what part of the representation is context-dependent. We finally compare the verb type vectors learned by the parser to verb type vectors learned with a language modeling objective. We expect the vectors learned by the parser to encode information about agreement and transitivity to a greater extent than the vectors learned using a language modelling objective. I explain this in more detail in Section 5.3.3.

Collecting FMVs such as in Figure 5.5 in UD treebanks is straightforward: verbs are annotated with a feature called *VerbForm* which has the value *Fin* if the verb is finite. We find candidates using the feature *VerbForm=Fin* and only keep those that are not dependent of a copula or an auxiliary dependency relation to make sure they are not part of a larger verbal construction.

We can collect AVCs in the UD and in the MS training data using the first part of the transformation and backtransformation algorithms respectively which I defined in Section 5.2.1. We scan the sentence left-to-right, looking for auxiliary dependency relations and collecting information about what is the outermost auxiliary and what is the main verb. When we have our set of FMVs and AVCs, we can create our task data sets.

The transitivity task is a binary decision of whether the main verb has an object or not. This information can be obtained by looking at whether or not the main verb has an *obj* dependent. In UD, a verb can have only one such dependent.[6] For the agreement task, we look at the morphological features of the verbs (the FMV or the auxiliary in case of AVCs) and concatenate the features *Person* and *Number*. The possible values are therefore all possible combination of 1st, 2nd and 3rd person with plural and singular. There are cases where this information is not available, in which case the agreement task is undefined for the AVC.

We release the code for this set of experiments,[7] including the modifications we made to the parser to freeze the vector representations at the different layers in the network.

5.3.2 Vector Representations

As described in Section 5.3.1, we are interested in two types of vectors: the vectors which are used for making parsing decisions, these are the vectors v_i at the output of the BiLSTM (Equation 5.1). These vectors represent the word in the context of the sentence, the *token*. We are also interested in the vectors representing the word *type* x_i (Equation 5.2), which is composed of an embedding of the word form $e(w_i)$ and a vector representing the character sequence of the word $ce(w_i)$. We refer to these as the *token*, *type* and *character* vectors respectively in this chapter. AVCs are represented by the *token* vectors of their head: the main verb in the UD representation and the outermost auxiliary in the MS representation.

$$v_i = BiLSTM(x_{1:n}, i) \tag{5.1}$$

$$x_i = [e(w_i); ce(w_i)] \tag{5.2}$$

These vectors are obtained by a BiLSTM which is a sequential model and does not explicitly learn hierarchical structure. However, as explained in different parts of this thesis, some work has suggested that LSTMs are capable of learning hierarchical structure which means that these vectors may contain information about their subtree. Our experiments from Chapter 4 have provided some evidence that BiLSTMs are indeed powerful feature extractors that make the use of a recursive composition function superfluous, indicating that token vectors probably contain information about their subtree. However, we also hypothesised that the advantages and disadvantages of structural modelling cancel each other out in our parser when we only look at parsing accuracy.

[6]We experimented with a harder task: predicting the number of objects. In the example in Figure 5.6, that number would be 1. In case of intransitive use of verbs, it would be 0, and with a ditransitive use of a verb (indirect objects are *iobj* dependents of the verb), it would be 2. We observed the same trends and therefore do not report those results.

[7]`https://github.com/mdelhoneux/avc_analyser`

It seems interesting to find out whether or not a composition function can be useful to model the specific case of transfer relations such as is found in AVCs. For this reason, we also investigate the composed representation of AVCs.

We are therefore interested in finding out whether or not an LSTM trained with a parsing objective can learn the notion of dissociated nucleus as well as whether or not a recursive composition function can help to learn this notion. As just mentioned, the head of an AVC in UD is a non-finite main verb, which we will refer to as NFMV, as depicted in Figure 5.6. The head of an AVC in MS is the outermost auxiliary, which we will refer to as the main auxiliary MAUX, as also depicted in that figure. We therefore look at NFMV and MAUX token vectors for the respective representation schemes and consider two definitions of these: One where we use the BiLSTM encoding of the main verb token v_i, and the other where we construct a subtree vector c_i^t by recursively composing the representation of AVCs as auxiliaries get attached to their main verb, as in Equation 4.5, repeated in Equation 5.3. When training the parser, we concatenate this composed vector to a vector of the head of the subtree to form v_i. In Chapter 4, we used two different composition functions, one using a simple recurrent cell and one using an LSTM cell. We saw that the one using an LSTM cell performed better. However, in this set of experiments, we only do recursive compositions over a limited part of the subtree: only between auxiliaries and NFMVs. This means that the LSTM would only pass through two states in most cases, and never more than 4.[8] This does not allow us to learn proper weights for the input, output and forget gates. An RNN cell seems more appropriate here and we only use that.

$$c_i^t = tanh(W[c_i^{t-1}; d; r] + b) \tag{5.3}$$

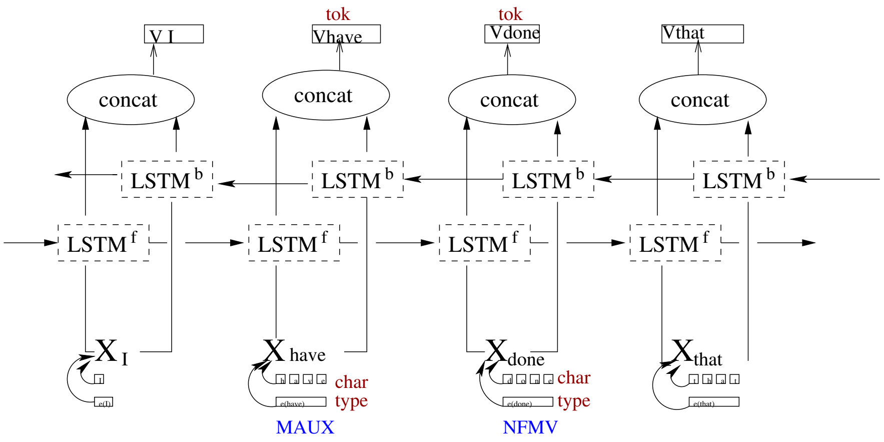


Figure 5.7. Example AVC with vectors of interest.

[8]The maximum number of auxiliaries in one AVC in our dataset is 3.

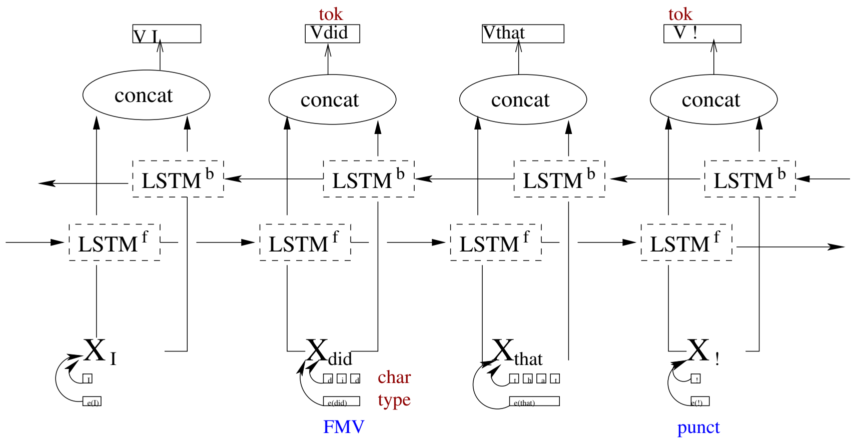


Figure 5.8. Example sentence with a FMV with vectors of interest.

I illustrate the vectors of interest with in Figure 5.7 and Figure 5.8, except for the word2vec ones and the ones obtained recursively.

5.3.3 Research Questions

As mentioned at the beginning of this section, our main research question is the question of whether or not a BiLSTM-based parser learns the notion of dissociated nucleus. We investigate this by looking at transitivity and agreement tasks. We first need to verify that agreement and transitivity information is learned by the parser and specifically, that this information is available to the parser when making decisions about main verbs. Our research questions can be formulated as follows, with the first being a pre-condition for the others:

RQ3a Is information about agreement and transitivity learned by the parser?
RQ3b Does a transition-based parser using a BiLSTM for feature extraction learn the notion of dissociated nucleus?
RQ3c Does a transition-based parser using a BiLSTM and augmented with a composition function over the AVC elements learn the notion of dissociated nucleus?

RQ3a

We verify that the parser has information about transitivity and agreement available when making parsing decisions about FMVs by comparing the accuracy of classifiers trained on token vectors of FMVs on these tasks to the majority baseline. We expect them to perform substantially better than that.

To verify that the BiLSTM does not propagate the information about agreement and transitivity throughout the network and that it is when making parsing decisions about main verbs that this information is available, we look at whether or not this information is encoded in punctuation marks: we collect the closest punctuation item that attaches to the FMV wherever possible. We expect these vectors to be uninformative about the tasks.

If this information is available in token vectors of FMVs, we are also interested in finding out how this information was obtained from the network. If it is available in the context-independent representation of words, i.e. the type and character representation of the word, it may be propagated upwards from these representations to the token representation. Otherwise, we know that it is learned by the BiLSTM.

If it is present in the context-independent representation of words, there is some indication that this information is learned by the parser. To verify that it is learned specifically for the task of parsing, we compare type vectors of FMVs obtained using our parser to vectors trained using a language modelling objective. We train a word2vec (Mikolov et al., 2013c) model on the same training set as for parsing. We thus obtain vectors of the same dimension as our word type vectors and trained with the same data set but trained with a different objective.[9]

To sum up, we expect the following to hold:

- Main verb token vectors are informative with respect to transitivity and agreement: they perform better than the majority baseline on these tasks.
- Main verb token vectors are more informative than punctuation token vectors with respect to transitivity and agreement.
- Main verb type vectors trained with a parsing objective contain more information about transitivity and agreement than type vectors trained with a language modelling objective.

RQ3b and RQ3c

If the parser learns a notion of dissociated nucleus, we expect to observe that AVC subtree vectors (i.e. the AVC's NFMV token vector or its composed version for UD, the AVC's MAUX token vector or its composed version for MS) contain a similar amount of information about agreement and transitivity as FMVs do.

[9]Note that for this kind of experiment, training language models that learn contextual representations of words such as ELMo (Peters et al., 2018) or BERT (Devlin et al., 2019) and comparing these representations to our token vectors would be more appropriate. However, these models were not available at the time of running our experiments. In addition, these models are typically trained on very large datasets and it is unclear how well they perform when trained on just treebank data. We leave this to future work.

5.3.4 Experiments

Data

To select our dataset, we want to follow as much as possible the criteria that we defined in Chapter 3. These are repeated below:

1. Typological variety
   a) Only different genera and as many families as possible
   b) Diversity in morphological complexity
   c) One treebank with high non-projective arcs ratio
2. Variety in treebank sizes and domains
3. High annotation quality

One of these criteria is not relevant here: the selection of a treebank with a high ratio of non-projectivity. Since we are analysing what a network learns, and not analysing the model's performance, this is not a concern. We have additional criteria relevant to this set of experiments. We need to make sure to have enough AVC examples, we need to make sure that the AVCs are annotated as correctly as possible, and we need to make sure that the treebanks contain the information we need for the prediction tasks. These added criteria are summarised as such:

4. A minimum amount of AVCs (at least 4,000)
5. High annotation quality of AVCs
6. Availability of the information we need for the prediction tasks

As mentioned in Chapter 3, there is an official evaluation script to collect all instances of known issues in treebanks, and we can look at the quality of auxiliary chains to know if AVCs are well annotated.[10] The information we need for the prediction tasks are: the presence of *Verbform=Fin* in morphological features, so as to collect FMVs, and the presence of the Number and Person features for the agreement task. These added criteria make the selection more difficult, as they exclude a lot of the small treebanks and make it difficult to keep typological variety, since the bigger treebanks come from the same language families. We select Catalan (AnCora), Croatian (SET), Dutch (Alpino) and Finnish (TDT). This selection satisfies our criterion to select languages from different genera. It also satisfies the criterion to have as many language families as possible: although we only have 2, it would have been difficult to have more because of the other criteria. The criterion to have languages with diverse morphological complexity is not perfectly met: we have no isolating language. However, there is no isolating language that fits the other criteria, so we see no other choice than relaxing that criterion for this study. Table 5.3 summarises the data used. We use UD version 2.2 (Nivre et al., 2018).

[10]https://github.com/UniversalDependencies/tools/blob/master/validate.py

Table 5.3. *Dataset sizes for the transitivity (T) and agreement (A) tasks for finite verbs and AVCs (FMV and AVC) as well as punctuation marks.*

		FMV		punct		AVC	
		train	dev	train	dev	train	dev
T	Catalan	14K	2K	7K	964	12K	2K
	Croatian	6K	803	4K	491	5K	653
	Dutch	9K	618	6K	516	5K	251
	Finnish	12K	1K	9K	1K	4K	458
A	Catalan	14K	2K	7K	964	12K	2K
	Croatian	6K	803	4K	491	5K	653
	Dutch	9K	618	6K	516	5K	246
	Finnish	10K	1K	8K	850	4K	443

Table 5.4. *LAS results of the baseline (bas) and recursive (rc) parser with UD and MS representations..*

	ud_bas	ud_rc	ms_bas	ms_rc
Catalan	87.8	87.7	87.8	87.8
Croatian	81.0	80.8	80.5	80.6
Dutch	84.1	83.7	84.1	83.7
Finnish	78.9	78.9	78.6	78.7
Average	83.0	82.8	82.7	82.7

Vectors

We train parsers for 30 epochs and pick the model of the best epoch based on LAS score on the development set. We report parsing results in Table 5.4 (some of these results are repeated from Section 5.2.) Note that these models correspond to the UD2UD and the MS2MS models as defined in Section 5.2.

We use the hyperparameters from the previous section (Table 5.1). Token vectors have a dimension of 250, type vectors 100 and character vectors 50. We load the parameters of the trained parser, run the BiLSTM through the training and development sentences, and collect the token vectors of NFMVs or MAUX, as well as the type and token vectors of FMVs. We use the vectors collected from the training set to train the classifiers and test on the vectors of the development sets.

As punctuation vectors, we take the token vectors of the punctuation marks that are closest to the FMV. We look at children of the FMV that have *punct* as dependency relation and take the closest one in linear order, first looking at the right children and then at the left ones.

As for word2vec vectors, we use the Gensim (Řehůřek and Sojka, 2010) implementation with default settings: using CBOW, a window of 5 and ignoring words with lower frequency than 5. We learn embeddings of the same dimension as our type vectors for comparability: 100.

In the *recursive* setup, we load the model, pass the data through the BiLSTM and parse the sentences so as to obtain composed representations of AVCs. For that, we collect the final composed vectors of NFMVs or MAUX after prediction for each sentence.

Results
We compare prediction accuracies to majority baselines. To get a measure that is comparable across languages and settings, we compute the difference between the accuracy of a classifier on a task using a set of vectors to the majority baseline for this set of vectors. The larger the difference, the greater the indication that the vector encodes the information needed for the task.[11] We perform paired t-tests to measure whether the difference to the majority baseline is statistically significant on average. Given that the set of labels we predict is very restricted, the majority baseline is reasonably good. As can be seen in Table 5.5 and Table 5.6, the accuracies ranges from 49 to 81 with most values around 60. It seems reasonable to assume that a system which performs significantly better than that learns information relevant to the notion predicted. The majority baselines of FMVs and AVCs are on average very close (Table 5.6), indicating that results are comparable for these two sets.[12]

RQ3a: Is agreement and transitivity information learned by the parser?

Results pertaining to **RQ3a** are given in Table 5.5. Note that we only report results on the UD data here since the representation of finite verbs does not change between UD and MS. The results conform to our expectations. FMV token vectors contain information about both agreement and transitivity. They perform significantly better than the majority baseline for both tasks.

Token vectors of punctuation marks related to FMVs contain some information about transitivity, with some variance depending on the language and perform significantly better than the majority baseline. However, they perform considerably worse than token vectors of FMVs. The difference between FMV token vectors and their majority baseline is substantially larger than the difference between punctuation vectors and their majority baseline for all languages and with 17 percentage points more on average. Punctuation vectors seem to be completely uninformative about agreement. The classifier seems to learn the majority baseline in most cases. This indicates that information about agreement and transitivity is relevant for scoring transitions involving FMVs but it is not necessary to score transitions involving tokens that are related to them, at least it is not necessary for related punctuation tokens. It indicates that this information is available in contextual information of FMVs but not in contextual information of all tokens in the verb phrase, except marginally for transitivity.

[11]We calculated relative error reductions as well but since these results showed the same trends, we exclusively report absolute difference in accuracy.

[12]The colour scheme in the tables indicates no change (yellow) to improvements (dark green).

Table 5.5. *Classification accuracy of the majority baseline (maj) and classifier trained on the type, token (tok) and word2vec (w2v) vectors of FMVs, and token vectors of punctuation (punct) on the agreement (A) and transitivity (T) tasks. Difference (δ) to majority baseline of these classifiers. Average differences significantly higher than the baseline with $p < .05$ are in bold and with $p < .01$ are in bold and italics.*

		FMV					punct		δ FMV				δ punct
		maj	tok	type	char	w2v	maj	tok	tok	type	char	w2v	tok
T	ca	70.5	88.7	79.4	75.0	74.4	67.5	71.3	18.2	9.0	4.5	3.9	3.8
	hr	55.9	79.7	71.3	70.6	57.8	61.5	62.7	23.8	15.4	14.7	1.8	1.2
	nl	61.7	82.1	74.0	69.4	64.8	62.0	69.6	20.5	12.4	7.7	3.1	7.6
	fi	59.2	86.2	72.8	74.9	59.2	56.6	64.1	27.0	13.5	15.6	0.0	7.5
	av	61.8	84.2	74.4	72.5	64.0	61.9	66.9	***22.4***	**12.6**	**10.7**	2.2	**5.0**
	sd	6.2	4.0	3.5	2.9	7.5	4.5	4.2	3.9	2.7	5.4	1.7	3.1
A	ca	74.4	82.2	82.6	98.4	74.4	76.7	76.6	7.7	8.1	24.0	0.0	-0.1
	hr	60.9	78.1	74.8	97.8	60.9	64.0	64.0	17.2	13.9	36.9	0.0	0.0
	nl	81.6	87.2	85.7	96.3	81.6	81.8	80.5	5.7	4.2	14.8	0.0	-1.2
	fi	61.6	86.0	63.2	93.5	61.6	59.7	59.7	24.4	1.6	31.9	0.0	0.0
	av	69.6	83.4	76.6	96.5	69.6	70.5	70.2	**13.7**	**6.9**	***26.9***	0.0	-0.3
	sd	10.1	4.1	10.0	2.2	10.1	10.4	10.0	8.7	5.4	9.7	0.0	0.6

We can conclude that information about both agreement and transitivity are learned by the parser. We can now look more closely at where in the network this information is present by looking at context independent vectors: type and character vectors. FMV type vectors seem to encode some information about transitivity: on average, they perform significantly better than the majority baseline on the transitivity task. When it comes to agreement, the difference to the baseline for FMV type vectors is smaller but they still perform significantly better than the majority baseline on average, although with more variation: they seem uninformative for Finnish. FMV token vectors contain substantially more information than type vectors for both tasks. The difference between token vectors and their majority baseline is, in most cases, slightly to substantially larger than it is between type vectors and their majority baseline, but on average, it is substantially larger.

FMV token vectors also contain substantially more information than character vectors for the transitivity task. For the agreement task, however, character vectors contain substantially more information than token vectors.

This indicates that the information flow for the two tasks differs to some extent. Since token vectors contain much more information about transitivity than both type and character vectors, we can conclude that this information is obtained from the BiLSTM.[13] For agreement, however, token vectors are less informative than character vectors which indicates that part of this information comes from the character vector but some of it gets filtered out by the BiLSTM. This indicates that agreement may be useful for signalling potential relationships between words which are then captured by the BiLSTM. A substantial part of the information remains though, indicating that agreement information is still useful when it comes to making parsing decisions.

We finally compare representations of word types when trained for the parsing as opposed to the language modelling task. For this, we look at word2vec vectors. Word2vec vectors of FMVs contain little information about transitivity, from no difference with the majority baseline to 3.9 percentage points above it. FMV type vectors are substantially better than word2vec vectors for all languages, with an average difference to the majority baseline that is 10 percentage points larger than the difference between word2vec vectors and their majority baseline. Word2vec vectors contain no information at all about agreement; the network learns the majority baseline for these vectors for all languages. FMV type vectors are better on this task for all languages. The difference between FMV type vectors and the majority baseline is small for Finnish, but on average, it is 6.9 percentage points better than the majority baseline. This indicates that information about transitivity and agreement is more relevant for the task of parsing than for the task of language modelling.

We have clearly seen 1) that transitivity and agreement are learned by the parser and that some information about these tasks is available to the parser when making decisions about FMVs and 2) that this information is not available everywhere in the network and is therefore available specifically when making decisions about FMVs. This answers **RQ3a** positively.

We should keep in mind that we observed a different information flow for transitivity where information is obtained mostly by the BiLSTM compared to agreement where it seems to be strongly signalled at the layer of context independent representations (in particular in the character vector) and weaker at the output of the BiLSTM.

RQ3b: Does a BiLSTM-based parser learn the notion of dissociated nucleus?

Results pertaining to **RQ3b** are given in Table 5.6. Comparing first FMV and NFMV token vectors, we can see that NFMVs are somewhat better than FMVs at the transitivity task but both perform substantially better than the majority baseline. On the agreement task, however, FMVs vectors perform substantially better than NFMV vectors. NFMV vectors in a UD representation seem completely uninformative when it comes to agreement, performing on average

[13]Further evidence for this conclusion comes from the fact that some of this information, unlike the agreement information, seems to spill over on neighboring tokens like the punctuation tokens considered earlier.

Table 5.6. *Classification accuracy of the majority baseline (maj) and classifier trained on the token (tok), type and character (char) vectors of FMVs, NFMVs and MAUX on agreement (A) and transitivity (T) tasks. Difference (δ) to majority baseline of these classifiers. Average differences significantly higher than the baseline with $p < .05$ are in bold and with $p < .01$ are in bold and italics.*

		FMV		AVC	NFMV-UD			MAUX-MS			FMV	δ nfmv			δ maux		
		maj	tok	maj	tok	type	char	tok	type	char	tok	tok	type	char	tok	type	char
T	ca	70.5	88.7	66.9	89.3	78.8	74.3	88.5	66.8	66.7	18.2	22.4	11.9	7.4	21.7	-0.1	-0.2
	hr	55.9	79.7	51.2	82.5	75.4	69.7	74.9	55.8	56.1	23.8	31.4	24.3	18.5	23.7	4.7	4.9
	nl	61.7	82.1	70.5	88.6	74.5	71.5	86.8	80.2	81.6	20.5	18.1	4.0	1.0	16.3	9.6	11.1
	fi	59.2	86.2	49.1	81.8	70.7	71.1	72.4	62.0	61.0	27.0	32.7	21.5	21.9	23.3	12.8	11.8
	av	61.8	84.2	59.4	85.6	74.8	71.6	80.6	66.2	66.3	***22.4***	***26.1***	**15.4**	**12.2**	***21.2***	**6.8**	**6.9**
	sd	6.2	4.0	10.8	3.9	3.3	1.9	8.2	10.3	11.1	3.9	7.0	9.3	9.7	3.4	5.7	5.7
A	ca	74.4	82.2	76.0	76.0	76.0	76.0	83.3	99.3	99.7	7.7	0.0	0.0	0.0	7.2	23.2	23.7
	hr	60.9	78.1	71.2	71.2	71.9	71.2	77.2	95.3	94.6	17.2	0.0	0.7	0.0	6.0	24.0	23.4
	nl	81.6	87.2	72.4	70.4	72.4	72.4	89.4	99.5	100.0	5.7	-2.0	0.1	0.0	17.1	27.2	27.6
	fi	61.6	86.0	67.5	68.5	67.5	69.8	77.7	89.6	92.3	24.4	1.0	0.0	2.3	10.2	22.1	24.8
	av	69.6	83.4	71.8	71.5	72.0	72.3	81.9	95.9	96.7	**13.7**	-0.2	0.2	0.6	**10.1**	***24.1***	***24.9***
	sd	10.1	4.1	3.5	3.2	3.5	2.7	5.7	4.6	3.8	8.7	1.2	0.3	1.2	5.0	2.2	1.9

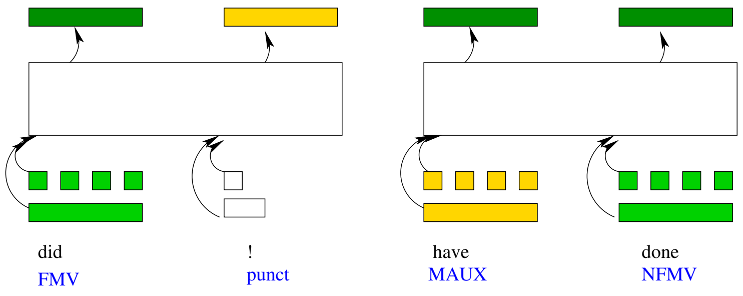
slightly worse (-0.2 percentage points) than the majority baseline. FMV vectors perform slightly (Dutch) to largely better (Finnish) than this depending on the language, with an average difference to the majority baseline of 13.7.

An unpaired t-test reveals that the results of FMVs and NFMVs on the agreement task are significantly different with $p < .01$, which further supports the hypothesis that they do not capture the same information. The results between FMVs and NFMVs are not significantly different on the transitivity task.

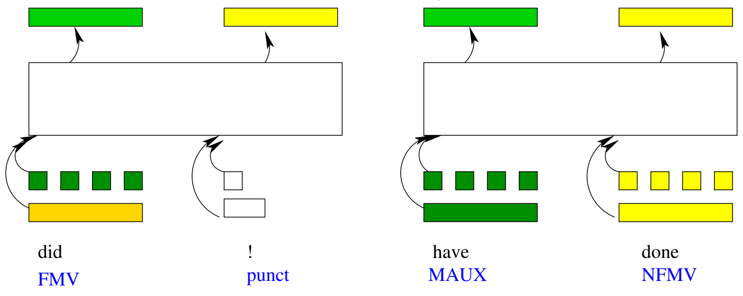
Looking at MAUX token vectors in an MS representation, we see that they are also substantially better than the majority baseline on the transitivity task, performing slightly worse than FMVs. Contrary to NFMV token vectors, they perform significantly better than the majority baseline on the agreement task and somewhat worse than FMVs. The results between MAUX and FMVs are not significantly different for either of the tasks, indicating that they do seem to capture a similar amount of information.

We can conclude that a BiLSTM-based parser does not learn the notion of dissociated nucleus for AVCs when working with a representation of AVCs where the main verb is the head such as is the case in UD: the representation of NFMVs contains less information about agreement than the representation of FMVs. However, when using a representation where the auxiliary is the head, a BiLSTM-based parser does seem to learn this notion; it learns a representation of the AVC's head that is similar to the representation of FMVs.

This can be explained by the different information flow in the network for the two tasks. In Figure 5.9a and 5.9b, I illustrate further the different information flow for the transitivity and agreement task respectively and for both



(a) Transitivity.



(b) Agreement.

Figure 5.9. Information flow for transitivity and agreement.

FMVs and AVCs. I use the same colour scheme as in the tables (from yellow to dark green means no information about a task to a substantial amount of information about the task, as measured by the difference to the majority baseline) and I simplify the architecture illustration from Figure 5.7 and 5.8. As we saw in Table 5.5 and as we can see from the left part of these figures, looking at FMVs, information about transitivity is mostly obtained from the BiLSTM whereas information about agreement is present in the character vector and propagated to the token vector. We observe a similar phenomenon with AVCs, as presented in Table 5.6: for the transitivity task, the type vectors of NFMVs contain more information than the type vectors of MAUX but in both cases, the token representation of the head of the AVC contains substantially more information than the type and character vectors. By contrast, both type and character vectors of MAUX contain information about agreement, whereas NFMV type and character vectors do not. It seems that, with this model, in order for agreement information to be available to the head of the AVC, the head of the AVC needs to be the auxiliary. When it comes to transitivity, the BiLSTM is able to construct this information regardless of what word is the head of the AVC, which is why the model is able to learn the notion of dissociated nucleus for the MS representation but not the UD representation.

Since the MS representation does not improve parsing accuracy (Table 5.4 and Table 5.2), it is possible that either learning this notion is not important for parsing, or that the benefits of learning this notion are offset by other factors. We attempt to find out whether we can get the best of both worlds by training a BiLSTM parser that uses composition on the UD representation. We also look at what happens with composition with the MS representation.

Note that, in all experiments, I only report results for NFMVs with a UD representation and for MAUX with an MS representation because these are the vectors that represent the subtree. However, the information that is learned in these vectors does not seem to depend much on the representation of the AVC: token, type and character vectors representing MAUX learn mostly similar amount of information about the two tasks whether in UD or in MS and token, type and character vectors of NFMVs also learn similar amounts of information whether in UD or in MS. This can be seen in in Appendix D where I report results with a MS representation of NFMVs and a UD representation of MAUX. Consequently, the parser learns similar representations of AVC elements and only the representation of the subtree depends on the representation style of AVCs.

RQ3c: Does subtree composition help?

As described at length in this thesis, there is evidence in the literature that it is unnecessary to use recursive neural networks to model hierarchical structure, as recurrent LSTMs seem to be able to capture hierarchical structure. We saw in Chapter 4 that composition does not make our parsing model more accurate. Composition might not be necessary for parsing accuracy but might help in this case, however. It could make it possible to get the relevant information from the main verb and the auxiliary token vectors. As we have just seen, the token vector of the MAUX has information that is missing in the token vector of the NFMV and that could be propagated to the NFMV vector through the recursive composition function.

We train a version of the parser where we recursively compose the representation of AVC subtrees. For UD, this means that the representation of the NFMV token gets updated as auxiliaries get attached to it. For MS, this means that the representation of AVC subtrees is composed in a chain from the outermost auxiliary to the main verb. Note that the composition function models the transfer relation for UD: it is used when an auxiliary is attached to the main verb. In MS, by contrast, it can also be used between two auxiliaries. This decreases parsing accuracy very slightly (Table 5.4). We compare the vectors of this composed representation to the representation of FMVs, see Table 5.7.

On average, composed NFMV vectors perform similarly to non-composed NFMV vectors on the transitivity task, slightly worse but substantially better than the majority baseline. For the agreement task, composed NFMV vectors are much better than the non-composed NFMV vectors, all performing substantially better than the majority baseline, although with less variation than the FMV vectors. The difference between composed NFMV vectors and the majority baseline is slightly higher (0.7 percentage points) than the difference

		FMV		AVC	NFMV		MAUX		δ				
		maj	tok	maj	tok	tok+c	tok	tok+c	fmv	nfmv	nfmv+c	maux	maux+c
T	ca	70.5	88.7	66.9	89.3	88.5	88.5	86.5	18.2	22.4	21.7	21.7	19.7
	hr	55.9	79.7	51.2	82.5	82.0	74.9	79.5	23.8	31.4	30.9	23.7	28.3
	nl	61.7	82.1	70.5	88.6	83.4	86.8	88.5	20.5	18.1	12.9	16.3	17.9
	fi	59.2	86.2	49.1	81.8	76.0	72.4	79.4	27.0	32.7	26.9	23.3	30.3
	av	61.8	84.2	59.4	85.6	82.5	80.6	83.5	***22.4***	***26.1***	***23.1***	***21.2***	***24.0***
	sd	6.2	4.0	10.8	3.9	5.2	8.2	4.7	3.9	7.0	7.8	3.4	6.2
A	ca	74.4	82.2	76.0	76.0	91.6	83.3	77.1	7.7	0.0	15.6	7.2	1.1
	hr	60.9	78.1	71.2	71.2	83.6	77.2	73.3	17.2	0.0	12.4	6.0	2.1
	nl	81.6	87.2	72.4	70.4	84.0	89.4	83.6	5.7	-2.0	11.6	17.1	11.2
	fi	61.6	86.0	67.5	68.5	79.2	77.7	74.7	24.4	1.0	11.7	10.2	7.2
	av	69.6	83.4	71.8	71.5	84.6	81.9	77.2	**13.7**	-0.2	**12.8**	**10.1**	5.4
	sd	10.1	4.1	3.5	3.2	5.2	5.7	4.6	8.7	1.2	1.9	5.0	4.7

Table 5.7. *Classification accuracy of the majority baseline (maj) and classifier trained on the vectors (vec) of NFMVs with (+c) and without composition and FMVs on agreement (A) and transitivity (T) tasks. Difference (δ) to majority baseline of these classifiers. Average differences significantly higher than the baseline with $p < .05$ are in italics and with $** = p < .01$ are in bold.*

between FMV vectors and their majority baseline for transitivity and slightly lower for agreement, but with variation across languages. On average, they seem to capture a similar amount of information. An unpaired t-test reveals that there is no significant difference between the results of FMVs and composed NFMV vectors. We can therefore conclude that a recursive composition function on top of a BiLSTM allows the model to capture a similar amount of information about AVCs and their non-dissociated counterpart, FMVs. This indicates that composing subtree representations makes it possible for the parser to learn the notion of dissociated nucleus with a representation of AVCs where the head is the main verb.

With the MS representation, composition improves accuracy on transitivity (except for Catalan) but decreases accuracy on agreement, making it on average not significantly better than the majority baseline and making the MAUX representation almost completely uninformative with regards to agreement for Catalan and Croatian. There is, however, no statistical difference between the results of FMVs and composed MAUX on either of the tasks, indicating that they do capture a similar amount of information.

Overall, it seems that using a UD representation and a recursive composition function is the best option we have to have an accurate parser which captures the notion of dissociated nucleus, given our definition of what it means to capture this notion (that FMV and NFMV representations encode a similar amount of information about agreement and transitivity). This does not improve overall parsing accuracy which means either that it is not important to capture this notion for parsing or that the benefits of doing so are offset by drawbacks of this method. It would be interesting to find out whether learning this information is important for downstream tasks, which we leave to future work.

5.4 Conclusion

This chapter focused on AVCs which are interesting for different reasons. First, the choice of representation taken by UD has been claimed to be suboptimal for parsing. Second, they are a typical case of what has been called a dissociated nucleus by Tesnière (1959) and are well attested typologically.

This chapter investigated the impact of two different representations of AVCs for parsing: UD, where the main verb is the head and MS, where the auxiliary is the head. We found that, in the case of a neural parser, the impact is smaller than was shown previously with pre-neural parsers, but that the UD representation is better than the MS representation.

This chapter then looked at what parsers learn about AVCs. In this thesis, I have tried to find out how we can build linguistic knowledge into our parsing model. In this chapter, we identified AVCs as a possible construction for which we can build more linguistically sophisticated knowledge into our parsers than has been done previously. We took a first step in that direction by doing an extensive analysis of what our parser learns about them. We asked the question: **RQ3**: *How can we investigate what neural models learn about language when trained for the task of dependency parsing?* More specifically, we investigated the question of whether or not our parser learns the notion of dissociated nucleus. We found that it learns this notion with an MS representation of AVCs. However, with a UD representation, it does not learn this notion when it is used without a recursive composition function. We explained this difference between UD and MS by looking at how the network learns information about AVC tokens. In the previous chapter, we found that a recursive composition function is superfluous with our parser when we look at parsing accuracy, but this chapter found that it is useful for learning the notion of dissociated nucleus. This suggests that we have perhaps not yet found the best way to model hierarchical structure into our architecture. One possibility for future work would be to use it only for the cases of dissociated nuclei, or transfer relations. The representation of the head of a dependency relation might be a good representation of its subtree but we might need to compose the representation of the subtree when the relation is not a relation of dependency but of transfer where both elements share head properties.

6. Polyglot Parsing

This chapter is about building polyglot parsing models where one model is trained for several languages. In Chapter 2, I described UD and the opportunities it opens up. One opportunity is to incorporate linguistic knowledge into our models while being aware of typological diversity. So far in this thesis, we have investigated various ways to do this. Another opportunity is to leverage data from multiple languages and improve parsing accuracy, especially for low-resource languages. This is the opportunity we try to take advantage of here. With this, we aim to answer **RQ4**: *How can we leverage information from multiple treebanks in different languages to improve parsing performance on the individual languages?*

I first briefly describe related work on polyglot parsing in Section 6.1. I then present work where we build models for related languages and see parsing accuracy gains from doing so in Section 6.2. I finally investigate different parameter sharing strategies in a restricted scenario where we build parsing models for pairs of languages in Section 6.3. We investigate this for related and unrelated language pairs.

6.1 Polyglot Parsing as Multi-Task Learning

In Section 2.3.3, I described different ways in which multilingual resources have been leveraged to improve parsing accuracy. I described that the creation of harmonised treebanks such as UD has made it possible to easily train parsers on treebanks in different languages: polyglot parsers. I described the work of Vilares et al. (2016), which simply used treebank concatenation and showed that this can lead to improved parsing accuracy, and Ammar et al. (2016) who build a more complex model to train a parser for eight languages. Ammar et al. (2016) modify the stack-LSTM parser which was described in Chapter 4 by introducing a language embedding le. The language embedding le is constructed by passing a vector l through an affine transform and a $tanh$ activation (Equation 6.1). They try variations for constructing l: one is simply a one-hot encoding of the language ID, another encodes word order properties and a last one uses typological properties from WALS (Dryer and Haspelmath, 2013). They find that a one-hot language embedding works best. We use something similar, as described in the following section.

$$le = tanh(Wl + b) \tag{6.1}$$

This line of work can be viewed as treating polyglot parsing as multi-task learning where some of the parameters are shared, and some are language-specific, as described in Chapter 2. In the context of cross-lingual parsing, Guo et al. (2016) explicitly refer to the use of multi-task learning (MTL) for cross-lingual parsing, where they propose a specific method to share parameters of a parser. They use the stack-LSTM parser described in Chapter 4 and share the parameters which they believe to be language-independent (for example POS embeddings, vector representations of the history of parsing actions, the LSTMs representing stack and buffer) and do not share the parameters which they believe should be language-specific (for example the character vectors, the character BiLSTM and the LSTM over the parsing actions). Similarly, in the context of monolingual parsing but with treebanks annotated with different styles and with a pre-neural parser, Johansson (2013) suggests to view multi-treebank parsing as a multi-task learning problem and constructs a partially shared feature representation across treebanks. MTL is not mentioned in the other studies I described but these can be seen as doing other variations of parameter sharing. In Vilares et al. (2016) and in Ammar et al. (2016), all parameters are shared but in the latter, language embeddings allow to learn language-specific information for some parameters. We can generalise the parameter strategies to three main strategies. In the first, we do not (✗) share the parameters which means that we create separate parameters for the different languages. In the second, we do hard sharing (✓) of the parameters which means that we share the parameters and pretend that the two languages are the same. The third is what we call *soft* sharing (**ID**) of the parameters: we use a language embedding to allow the model to influence the parameters with language-specific information. These strategies are summarised as such:

✗	creating a parameter set per language
✓	sharing the parameter set across languages
ID	sharing the parameter set across languages while constructing a language embedding to allow influencing the parameters with language-specific information

In Section 6.2, we try a simple variant of parameter sharing in our parsing model where we do hard sharing of most of the parameters and soft sharing of some parameters at the level of word representations: we introduce a language embedding and add it to the word representation. We exploit this for related languages for which we find that this method works best. In Section 6.3, we explore various combinations of parameter sharing strategies in a more restricted setup: focusing on language pairs with a small amount of data. We do this both for related and unrelated languages. As mentioned in Chapter 2, a lot of work in cross-lingual and polyglot parsing has been using gold POS tags as part of the input and this has been shown to overestimate the gains that can be obtained from using these methods. We use predicted POS tags in Section 6.2 where I present results obtained as part of the CoNLL shared task 2018. We do not use POS tags at all in Section 6.3. A question underlying this chapter is therefore the following:

RQIII Can we obtain gains from polyglot parsing without access to gold or predicted POS tags?

6.2 Polyglot Parsing for Related Languages

As mentioned in Section 6.1, inspired by the idea of using language embeddings from Ammar et al. (2016), we introduce something similar to a language embedding in our parsing model. Instead of representing the language, however, the embedding represents the treebank. We found that using treebank embeddings improves parsing accuracy even in the monolingual setting, when training a parser with heterogeneous treebanks in the same language, as described in Stymne et al. (2018). We simply build a lookup table with vectors representing treebanks seen in training. This is equivalent to using a language embedding with the language ID as one-hot vector such as is done in Ammar et al. (2016). We concatenate the treebank vector $te(w_i)$ of a word w_i to the representation of the word at the input of the BiLSTM. In its most complex form, therefore, the representation of word vectors are a concatenation of a pre-trained word embedding, an embedding representing the character sequence of the word, a POS embedding and a treebank embedding:

$$x_i = [pe(w_i); ce(w_i); e(t_i); te(w_i)] \tag{6.2}$$

This method can be used both for cross-lingual and polyglot parsing. As described in Section 2.3.3, in cross-lingual parsing we are working with languages that do not have equal status. There is a target language for which we have no or little data and for which we seek to build a model using one or multiple source languages for which we have more data. We first explored this idea in the CoNLL 2017 shared task with mixed success, only in a cross-lingual setting for the low-resource languages. Our method is described in Section 5 in de Lhoneux et al. (2017a) and experimental results in Section 6 in that paper. We obtained some encouraging results when using this method with related languages but less convincing results in other scenarios. Our results are encouraging since most work on cross-lingual and polyglot parsing has relied on POS tags, especially the delexicalised models. It is also encouraging because we do not use any external resources like in most cross-lingual work or in Ammar et al. (2016). Our method is only slightly more complex than the method of treebank concatenation proposed by Vilares et al. (2016). We therefore have indications that we can obtain improvements from polyglot parsing without the use of POS tag information or external resources.

We explored this idea more extensively in the CoNLL 2018 shared task, by training more polyglot models, which is what I describe in this section. With this, we attempt to answer the following question:

RQ4a Can we improve parsing accuracy with polyglot parsing compared to a monolingual baseline?

In the CoNLL 2018 shared task, we explore the idea of training multi-treebank and polyglot models much more extensively. Note that we also use POS tag embeddings here. The parser is part of the Uppsala system which was described in Section 3.4.2.

Parsing Models

We attempted to leverage resources from multiple treebanks as much as possible and therefore built multi-treebank models wherever possible. In this section, I describe our approach generally. The models are described in more details in Appendix E.

Treebanks sharing a single model are grouped together in Table 6.1. To decide which languages to combine in our polyglot models, we used two sources: knowledge about language families and language relatedness, and clusterings of treebank embeddings from training our parser with all available languages. We created clusterings by training single parser models with treebank embeddings for all treebanks with training data, capping the maximum number of sentences per treebank to 800. We then used Ward's method (Ward, 1963) to perform a hierarchical cluster analysis. We found that the most stable clusters were for closely related languages. There was also a tendency for treebanks containing classic languages (i.e., Ancient Greek, Gothic, Latin and Old Church Slavonic) to cluster together. One reason for these languages clustering together could be that several of the treebanks of these languages come from the same annotation projects: four come from PROIEL, and two from Perseus. Treebanks from these projects contain consistently annotated and at least partially parallel data, e.g., from the Bible. We found in Stymne et al. (2018) that the treebank embedding method makes it possible to learn differences in annotation style and this is most probably happening here. For the multi-treebank models, we performed preliminary experiments on development data investigating the effect of different groupings of languages. The main tendency we found was that it was better to use smaller groups of closely related languages rather than larger groups of slightly less related languages. For example, using polyglot models only for Galician–Portuguese and Spanish–Catalan was better than combining all Romance languages in a larger model, and combining Dutch–German–Afrikaans was better than also including English. In the case where we have a very low-resource language (around 30 sentences or less), we construct a polyglot model but with less related languages than for our other polyglot models. We believe this will still be stronger than a monolingual base-

Table 6.1. *LAS results (+ mono-treebank baseline), sentence and word segmentation and UPOS tagging. Treebanks sharing a parsing model grouped together; Confidence intervals for coloring:* 🟥 $< \mu - \sigma <$ 🟥 $< \mu - \text{SE} < \mu < \mu + \text{SE} <$ 🟩 $< \mu + \sigma <$ 🟩.

LANGUAGE	TREEBANK	LAS		SENTS	WORDS	UPOS
ARABIC	PADT	73.54	73.54	68.06	96.19	90.70
ARMENIAN	ARMTDP	23.90	23.90	57.44	93.20	75.39
BASQUE	BDT	78.12	78.12	100.00	100.00	96.05
BULGARIAN	BTB	88.69	88.69	95.58	99.92	98.85
BRETON	KEB	33.62	33.62	91.43	90.97	85.01
CHINESE	GSD	69.17	69.17	99.10	93.52	89.15
GREEK	GDT	86.39	86.39	91.92	99.69	97.26
HEBREW	HTB	67.72	67.72	100.00	90.98	80.26
HUNGARIAN	SZEGED	73.97	73.97	94.57	99.78	94.60
INDONESIAN	GSD	78.15	78.15	93.47	99.99	93.70
IRISH	IDT	68.14	68.14	94.90	99.60	91.55
JAPANESE	GSD	79.97	79.97	94.92	93.32	91.73
JAPANESE	MODERN	28.27	28.27	0.00	72.76	54.60
LATVIAN	LVTB	76.97	76.97	96.97	99.67	94.95
OLD FRENCH	SRCMF	78.71	78.71	59.15	100.00	95.48
ROMANIAN	RRT	84.33	84.33	95.81	99.74	97.46
THAI	PUD	4.86	4.86	11.69	69.93	33.75
VIETNAMESE	VTB	46.15	46.15	88.69	86.71	78.89
AFRIKAANS	AFRIBOOMS	79.47	78.89	99.65	99.37	96.28
DUTCH	ALPINO	83.58	81.73	89.04	99.62	95.78
	LASSYSMALL	82.25	79.59	73.62	99.87	96.18
GERMAN	GSD	75.48	75.15	79.36	99.37	94.02
ANCIENT GREEK	PERSEUS	65.17	62.95	98.93	99.97	92.40
	PROIEL	72.24	71.58	51.17	99.99	97.05
GOTHIC	PROIEL	63.40	60.58	31.97	100.00	93.43
LATIN	ITTB	83.00	82.55	94.54	99.99	98.34
	PERSEUS	58.32	49.86	98.41	100.00	88.73
	PROIEL	64.10	63.85	37.64	100.00	96.21
OLD CHURCH SLAVONIC	PROIEL	70.44	70.31	44.56	99.99	95.76
BURYAT	BDT	17.96	8.45	93.18	99.04	50.83
KAZAKH	KTB	31.93	23.85	94.21	97.40	61.72
TURKISH	IMST	61.34	61.77	96.63	97.80	93.72
UYGHUR	UDT	62.94	62.38	83.47	99.69	89.19
CATALAN	ANCORA	88.94	88.68	99.35	99.79	98.38
SPANISH	ANCORA	88.79	88.65	97.97	99.92	98.69
CROATIAN	SET	84.62	84.13	96.97	99.93	97.93
SERBIAN	SET	86.99	85.14	93.07	99.94	97.61
SLOVENIAN	SSJ	87.18	87.28	93.23	99.62	97.99
	SST	56.06	53.27	23.98	100.00	93.18
CZECH	CAC	89.49	88.94	100.00	99.94	99.17
	FICTREE	89.76	87.78	98.72	99.85	98.42
	PDT	88.15	88.09	92.29	99.96	99.07
	PUD	84.36	83.35	96.29	99.62	97.02
POLISH	LFG	93.14	92.85	99.74	99.91	98.57
	SZ	89.80	88.48	98.91	99.94	97.95
SLOVAK	SNK	86.34	83.80	88.11	99.98	96.57
UPPER SORBIAN	UFAL	28.85	2.70	73.40	95.15	58.91
DANISH	DDT	80.08	79.68	90.10	99.85	97.14
FAROESE	OFT	41.69	39.94	95.32	99.25	65.54
NORWEGIAN	BOKMAAL	88.30	87.68	95.13	99.84	98.04
	NYNORSK	87.40	86.23	92.09	99.94	97.57
	NYNORSKLIA	59.66	55.51	99.86	99.99	90.02
SWEDISH	LINES	80.53	78.33	85.17	99.99	96.64
	PUD	78.15	75.52	91.57	98.78	93.12
	TALBANKEN	84.26	83.29	96.45	99.96	97.45
ENGLISH	EWT	81.47	81.18	75.41	99.10	95.28
	GUM	81.28	79.23	81.16	99.71	94.67
	LINES	78.64	76.28	88.18	99.96	96.47
	PUD	84.09	83.67	97.02	99.69	95.23
ESTONIAN	EDT	81.09	81.47	92.16	99.96	97.16
FINNISH	FTB	84.19	83.12	87.91	99.98	96.30
	PUD	86.48	86.48	92.95	99.69	97.59
	TDT	84.33	84.24	91.12	99.78	97.06
NORTH SAAMI	GIELLA	64.85	64.14	98.27	99.32	90.44
FRENCH	GSD	85.61	85.16	95.40	99.30	96.86
	SEQUOIA	87.39	86.26	87.33	99.44	97.92
	SPOKEN	71.26	69.44	23.54	100.00	95.51
GALICIAN	CTG	78.41	78.27	96.46	98.01	95.80
	TREEGAL	72.67	70.16	82.97	97.90	93.25
PORTUGUESE	BOSQUE	84.41	84.27	90.89	99.00	95.90
HINDI	HDTB	89.37	89.23	99.02	100.00	97.44
URDU	UDTB	80.40	79.85	98.60	100.00	93.66
ITALIAN	ISDT	89.43	89.37	99.38	99.55	97.79
	POSTWITA	76.75	76.46	54.00	99.04	95.61
KOREAN	GSD	81.92	81.12	92.78	99.87	95.61
	KAIST	84.98	84.74	100.00	100.00	95.21
KURMANJI	MG	29.54	7.61	90.85	96.97	61.33
PERSIAN	SERAJI	83.39	83.22	99.50	99.60	96.79
NAIJA	NSC	20.44	19.44	0.00	98.53	57.19
RUSSIAN	SYNTAGRUS	89.00	89.39	98.79	99.61	98.59
	TAIGA	65.49	59.32	66.40	97.81	89.32
UKRAINIAN	IU	82.70	81.41	93.42	99.76	96.89
ALL		72.37	70.71	83.80	98.18	90.91
BIG		80.25	79.61	87.23	99.10	95.59
PUD		72.27	71.46	75.57	94.11	87.51
SMALL		63.60	60.06	80.68	99.23	90.93
LOW-RESOURCE		25.87	18.26	67.50	93.38	61.07

line which is very low with such little amount of data. For example, for Buryat, we trained a model with Turkish, and for Kurmanji, we trained a model with Persian.

Results

Table 6.1 shows test results for the Uppsala system, including the main metrics LAS (plus a mono-treebank baseline for LAS), the sentence and word segmentation accuracy, and the accuracy of UPOS tagging. The rows ALL, BIG, PUD, SMALL and LOW-RESOURCE contains the macro-average of all or subsets of the test sets. The ALL category includes all treebanks. The BIG category includes the biggest treebanks, defined as the treebanks that have more training data than test and development data combined. The PUD category includes the parallel treebanks released for the CoNLL 2017 shared task. The SMALL category includes the small treebanks, defined as having a small but reasonable amount of training data and no development data. The LOW-RESOURCE category includes low-resource languages, defined as the treebanks that have no training data or a tiny sample of a few dozen sentences. To make the table more readable, we have added a simple colour coding scheme. Scores that are significantly higher/lower than the mean score of the 21 systems that successfully parsed all test sets are marked with two shades of green/red. The lighter shade marks differences that are outside the interval defined by the standard error of the mean ($\mu \pm \text{SE}, \text{SE} = \sigma/\sqrt{N}$) but within one standard deviation (std dev) from the mean. The darker shade marks differences that are more than one std dev above/below the mean ($\mu \pm \sigma$).[1]

We can see that our LAS scores are significantly above the mean for all aggregate sets of treebanks (ALL, BIG, PUD, SMALL, LOW-RESOURCE) with an especially strong result for the low-resource group, for many of which we trained polyglot models. If we look at specific languages, we do particularly well on low-resource languages like Breton, Buryat, Kazakh and Kurmanji, but also on languages like Arabic, Hebrew, Japanese and Chinese, where we benefit from having better word segmentation than most other systems. Our results are significantly worse than the mean only for Afrikaans AfriBooms, Old French SRCMF, Galician CTG, Latin PROIEL, and Portuguese Bosque. For Galician and Portuguese, this may be due to the effect of lower word segmentation and tagging accuracy.

To verify the usefulness of multi-treebank training, we also trained mono-treebank models so that we can compare parsing accuracy. We refer to this as the mono-treebank baseline, and the LAS scores can be found in the second (uncoloured) LAS column in Table 6.1. The results show that merging treebanks and languages does in fact improve parsing accuracy in a remarkably consistent fashion. For the 64 test sets parsed with a multi-treebank model,

[1]Note that this table contains official scores for the shared task which we found later violated a rule for Thai. The difference is small and has no impact on our ranking. Our results corrected to comply to the shared task rules can be found in Smith et al. (2018a).

only four have a (marginally) higher score with the mono-treebank baseline model: Estonian EDT, Russian SynTagRus, Slovenian SSJ, and Turkish IMST. Looking at the aggregate sets, we see that, as expected, the pooling of resources helps most for LOW-RESOURCE (25.33 vs. 17.72) and SMALL (63.60 vs. 60.06), but even for BIG there is some improvement (80.21 vs. 79.61). We verify our intuition that using less related languages for the very low-resource case does in fact improve over the monolingual baseline, see Buryat (17.96 vs. 8.45), Uyghur (62.94 vs. 62.38) and Kurmanji (29.54 vs. 7.61). We find these results very encouraging, as they indicate that our treebank embedding method is a reliable method for pooling training data both within and across languages. This answers **RQ4a** positively: we can improve over a monolingual baseline by training polyglot models. While in contrast with the previous shared task, we do make use of POS tags which may help with cross-lingual transfer, our system is still simpler than most work on cross-lingual parsing as it does not require external resources. It seems that hard parameter sharing of most of the network together with soft sharing at the word level is an efficient way of leveraging treebank data from multiple related languages and is beneficial in the low-resource scenario.

6.3 Parameter Sharing in Polyglot Parsing

As described in Section 6.1, polyglot dependency parsing can be viewed as multi-task learning where parsing in two different languages can be seen as two different tasks and therefore, where some parameters are shared across languages and some are language-specific. It was mentioned that Guo et al. (2016) conceptualised it like that and built a network where they made arbitrary decisions on which parameters to share. We did something similar in the previous section. In this section, we investigate different parameter sharing strategies, to find out which of the parameters it is best to share. We investigate this for related languages and unrelated languages for which we hypothesise that parameter sharing helps less but that sharing some of the parameters might still be helpful. We therefore attempt to answer the following questions:

RQ4b What parameters of a BiLSTM parser is it best to share for related languages?

RQ4c Can parameter sharing help parsing when working with unrelated languages?

RQ4d What parameters of a BiLSTM parser is it best to share for unrelated languages?

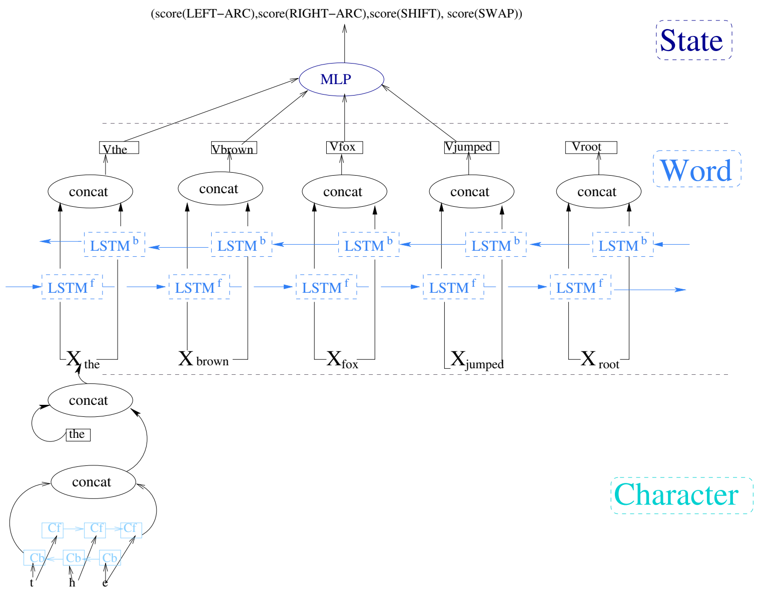


Figure 6.1. 3 sets of parameters in our parsing architecture.

6.3.1 Parameter Sharing Strategies

Our parsing architecture can be divided into three sets of parameters: the parameters of the characters (the character vectors and BiLSTM), the parameters of the words (the word vectors and BiLSTM) and the parameters of the configuration or state, the MLP classifier. This is illustrated in Figure 6.1.

In the setting where we do not share (✗) word parameters (**W**), we construct a different word lookup table and a different word-level BiLSTM for each language. In the setting where we do hard parameter sharing (✓) of word parameters, we only construct one lookup table and one word BiLSTM for the languages involved. In the setting where we do soft sharing (**ID**) of word parameters, we share those parameters, and in addition, concatenate a treebank or language embedding[2] $te(w_i)$ representing the language of word w_i to the vector of the word w_i at the input of the word BiLSTM:

$$x_i = [e(w_i); ce(w_i); te(w_i)] \tag{6.3}$$

Similarly for character parameters (**C**), we construct a different character BiLSTM and one character lookup for each language (✗), create those for all languages and share them (✓) or share them and concatenate an embedding representing the language of the word at the input of the character BiLSTM (**ID**):

$$ch_j = [e(ch_j); te(w_i)] \tag{6.4}$$

[2]Since we work with one treebank per language, treebank embeddings can be viewed as language embeddings in this section.

Table 6.2. *Hyperparameter values for parsing.*

Character embedding dimension	24
Character BiLSTM output dimension	50
Word embedding dimension	100
Word BiLSTM layers	2
Word BiLSTM hidden/output dimension	250
Treebank embedding dimension	12
Hidden units in MLP	100

At the level of configuration or parser states (**S**), we either construct a different MLP for each language (✗), share the MLP (✓) or share it and concatenate an embedding representing the language of the words of the configuration to the vector representing the configuration, at the input of the MLP (**ID**):

$$\phi(c) = [v_{s_1}; v_{s_0}; v_{b_0}; te(s_1, s_0, b_0)] \tag{6.5}$$

As mentioned in Chapter 3, we keep the architecture as simple as possible and avoid adding confounding factors, we restrict the word representation to a vector of the word form and a vector representing the characters of the words and we restrict the vector representing the configuration to the two top items of the stack and the first item of the buffer (those are the words that may be involved in a transition), as in Equation 6.5.

A flexible implementation of parameter strategies for UUParser using DyNet (Neubig et al., 2017).[3] is available online.[4]

6.3.2 Experiments

We train our parsers for 30 epochs with the hyperparameters in Table 6.2. We report the score of the best epoch on the development set.

Data

Due to the number of parameter sharing strategies to try, we investigate this in a restricted setup and we focus on pairs of languages. Following the arguments from Chapter 3, we make sure to include some typological variety in our language pair selection. We use UD version 2.1 (Nivre et al., 2017b). We focus on pairs of related languages (which we define as being from the same genus) since we have found, as presented in Section 6.2, that we can use polyglot parsing to improve parsing accuracy when training on related languages. We have not found clear evidence that this works with unrelated languages if they do not have the same annotation style (we found that multilingual training of parsers for classical languages annotated by the same project is useful.)

[3]https://github.com/clab/dynet

[4]https://github.com/coastalcph/uuparser

Table 6.3. *Dataset characteristics. IE means Indo-European.*

ISO	Lang	Tokens	Genus	Family	Word order
ar	Arabic	208,932	Semitic	Afro-Asiatic	VSO
he	Hebrew	161,685	Semitic	Afro-Asiatic	SVO
et	Estonian	60,393	Finnic	Uralic	SVO
fi	Finnish	67,258	Finnic	Uralic	SVO
hr	Croatian	109,965	Slavic	IE	SVO
ru	Russian	90,170	Slavic	IE	SVO
it	Italian	113,825	Romance	IE	SVO
es	Spanish	154,844	Romance	IE	SVO
nl	Dutch	75,796	Germanic	IE	No dom. order
no	Norwegian	76,622	Germanic	IE	SVO

This does not mean that we cannot obtain improvements from polyglot parsing using unrelated languages. However, we first focus on related languages, for which we know that polyglot parsing can work In a second step, we also look at pairs of unrelated languages (which we define minimally as being from different genera) to see what difference this makes. To make sure that our settings are comparable across language pairs and that we have balanced training sets, we cap the number of training sentences to 5,000. This is, according to us, the highest reasonable number that allows including a variety of languages, since most of the large treebanks are Indo-European and include only a few different genera. This setting is likely to overestimate the gains that we can get from polyglot parsing in the high-resource scenario for the high-resource languages, and underestimate the gains that can be obtained using more resources from high-resource languages for the low-resource languages. It should give us a good estimate of how much we can gain for low-resource languages when we only have access to low-resource languages which is a reasonable and interesting scenario. In this scenario, it is unrealistic to assume that we have access to an accurate POS tagger. This is an additional reason to refrain from using POS tags as part of the input. Since for some languages, it is hard to find a close relative, we choose pairs that are not too closely related, for example Norwegian and Swedish or Spanish and Portuguese would be too closely related. Finnish and Estonian might be an exception here; they are very closely related, although they are not considered mutually intelligible (Härmävaara, 2014). However, their inclusion allows us to have pairs from three different language families in total.

The dataset characteristics are listed in Table 6.3. Our treebank selection includes 10 languages, representing five language pairs from different genera. Our two Semitic languages are Arabic and Hebrew. These two languages differ in that Arabic tends to favour VSO word order whereas Hebrew tends to use SVO, but are similar in their rich transfixing morphology.[5] Our two Finno-Ugric languages are Estonian and Finnish. These two languages differ in that Estonian no longer has vowel harmony, but share a rich agglutinative morphology. Our two East and South Slavic languages are Croatian and Russian. These two languages differ in that Croatian uses gender in plural nouns, but otherwise share their rich inflectional morphology. Our two Romance languages are Italian and Spanish. These two languages differ in that Italian uses a possessive adjective with a definite article, but share a fairly strict SVO order. Finally, our two Germanic languages are Dutch and Norwegian. These two languages differ in morphological complexity, but share word ordering features to some extent.

Note that two pairs do not share a script: Arabic and Hebrew and Russian and Croatian. Sharing character representations therefore makes less sense than for the other languages without transliteration: the parameters of the character BiLSTM are shared but the character sets are different.

Results

We train parsers bilingually for our 5 language pairs with the 27 different parameter sharing strategies and compare the results to a monolingual baseline where a parser is trained separately for each language. Results are given in Table 6.4 where the parameter sharing strategies are ranked by average performance. These experiments are meant to answer **RQ4b**: what parameters is it best to share when training a polyglot parser for related languages? Our main observations are:

- Generally, and as observed in previous work, *multi-task learning helps*: all different sharing strategies are on average better than the monolingual baselines, with minor (0.16 LAS points) to major (0.86 LAS points) average improvements.
- Sharing the MLP seems to be overall a better strategy than not sharing it: the 10 best strategies share the MLP.
- Whereas the usefulness of sharing the MLP seems to be quite robust across language pairs, the usefulness of sharing word and character parameters seems more dependent on the language pairs.

These last two observation reflect the linguistic intuition that character- and word-level LSTMs are highly sensitive to phonological and morphosyntactic differences such as word order, whereas the MLP learns to predict less idiosyncratic, hierarchical relations from relatively abstract representations of

[5]A transfix is a discontinuous affix which occurs at more than one position in a word (`https://en.wiktionary.org/wiki/transfix`)

Table 6.4. *Performance (LAS; in %) across select sharing strategies ranked by average performance.* Mono *is our single-task baseline;* ***W*** *refers to sharing the word parameters,* ***C*** *refers to sharing the character parameters,* ***S*** *refers to sharing the MLP parameters.*

C	**W**	**S**	ar	he	es	it	et	fi	nl	no	hr	ru	av.
Mono			76.3	80.2	83.7	83.3	70.4	70.8	77.3	80.8	76.8	82.3	78.2
Language-best			76.6	80.6	84.4	84.8	72.8	72.9	79.6	82.1	78.0	82.9	79.5
✗	✓	ID	76.3	80.3	84.2	84.5	72.1	72.5	78.8	81.4	77.6	82.8	79.1
✗	✓	✓	76.4	80.4	84.1	84.4	71.9	72.0	78.7	81.5	**78.0**	82.8	79.0
✗	ID	ID	76.2	80.1	84.2	**84.8**	71.8	72.4	78.6	81.6	77.4	82.9	79.0
✗	ID	✓	76.4	80.4	84.2	84.3	72.2	72.3	78.3	81.6	77.4	82.8	79.0
✗	✗	ID	76.5	**80.6**	84.1	84.3	71.8	72.0	78.5	81.5	77.7	**82.9**	79.0
✓	ID	✓	76.2	79.8	**84.4**	84.7	72.4	71.5	79.2	81.6	76.9	82.8	78.9
ID	✗	✓	76.3	80.2	84.0	84.4	**72.8**	71.7	78.6	**82.1**	77.2	82.3	78.9
✗	✗	✓	**76.6**	80.3	84.0	83.7	71.5	**72.9**	78.3	81.5	77.4	82.8	78.9
ID	ID	ID	76.3	79.9	84.1	84.4	72.1	71.3	**79.6**	81.4	77.1	82.5	78.9
✓	ID	ID	76.2	80.1	84.3	84.5	72.5	71.8	78.7	81.1	76.7	82.6	78.8
ID	✗	✗	**76.6**	80.3	84.1	84.3	71.8	71.7	78.3	81.5	77.2	82.5	78.8
ID	✗	ID	76.2	79.9	84.1	84.3	71.9	72.0	78.5	81.7	77.2	82.3	78.8
✗	✗	✗	**76.6**	80.3	84.0	84.2	71.2	72.1	78.0	81.6	77.3	82.6	78.8
✓	✗	✓	76.1	80.1	84.0	84.1	72.1	72.0	78.2	81.8	76.8	82.6	78.8
✓	✗	ID	76.5	80.0	84.0	84.4	71.9	71.7	78.5	81.7	76.7	82.3	78.8
✓	✓	ID	**76.6**	80.1	84.0	84.6	72.1	71.0	78.3	81.0	76.9	82.8	78.7
ID	ID	✓	76.1	80.1	84.1	84.6	72.1	71.2	78.0	81.4	77.0	82.7	78.7
✓	✗	✗	76.4	80.3	84.3	84.0	72.3	71.0	78.3	81.3	77.0	82.3	78.7
✓	✓	✓	76.2	80.1	84.0	84.2	72.1	71.4	78.7	81.1	77.0	82.5	78.7
✗	ID	✗	**76.6**	80.1	84.1	84.3	71.7	71.6	77.6	81.0	77.0	82.6	78.7
ID	✓	✓	76.2	79.9	83.8	84.5	72.4	70.3	78.1	81.0	77.2	82.6	78.6
✗	✓	✗	76.3	79.9	83.9	84.4	72.4	71.3	77.4	80.7	76.9	82.5	78.6
ID	✓	ID	76.0	80.0	83.8	84.3	71.7	70.9	78.3	81.0	76.9	82.5	78.5
✓	✓	✗	76.1	79.7	83.8	84.5	71.9	70.4	77.8	81.1	76.5	82.3	78.4
✓	ID	✗	76.0	79.3	84.1	84.4	71.5	71.3	77.7	80.6	76.7	82.5	78.4
ID	✓	✗	76.1	79.9	83.9	84.4	71.1	70.6	77.8	80.9	77.0	82.0	78.4
ID	ID	✗	75.9	79.5	84.1	84.4	72.1	70.5	77.4	80.6	77.0	82.2	78.4

Table 6.5. *LAS on the test sets of the best of 9 sharing strategies and the monolingual baseline. δ is the difference between* Ours and Mono.

	W	**C**	Ours	Mono	δ
Arabic	✗	✗	77.2	77.1	0.1
Hebrew	✓	✗	80.0	79.8	0.3
Estonian	✗	ID	71.4	70.5	0.8
Finnish	✗	✗	71.6	71.6	0.1
Croatian	✓	✗	77.9	78.0	-0.1
Russian	✓	✗	83.5	82.7	0.8
Italian	ID	✓	85.0	84.0	1.0
Spanish	ID	✓	84.3	83.8	0.5
Dutch	ID	✓	75.5	74.1	1.4
Norwegian	✗	ID	81.1	80.1	1.0
av.			78.8	78.2	0.6

parser configurations. We note that sharing characters does not really help or harm languages with different scripts more than others. We also note generally that parameter sharing is more helpful for languages that are more related: the Finnish–Estonian pair gets the biggest gains and the Arabic–Hebrew pair gets the smallest gains from parameter sharing.

Based on this result, we propose a model (Ours) where *the MLP is shared (hard sharing) and the sharing of word and character parameters is controlled by a parameter that can be set on validation data*. We obtain a 0.6 LAS improvement on average and our proposed model is significantly better than the monolingual baseline with $p < 0.01$ (Table 6.5). Significance testing is performed using a randomisation test, with the script from the CoNLL 2017 Shared Task.[6]

We repeat the same set of experiments with the same languages but changing the pairs so that they are unrelated. We mix the pairs in such a way that the pairs are not only from a different genus, but also, as far as possible, from a different language family. Our pairs are therefore:

- Hebrew – Norwegian
- Finnish – Croatian
- Russian – Spanish

[6]`https://github.com/udapi/udapi-python/blob/master/udapi/block/eval/conll17.py`

- Italian – Estonian
- Dutch – Arabic

The only pair that violates this constraint is the pair with Russian and Spanish which are both Indo-European. This could not be avoided since there are 6 Indo-European languages in our selection of 10. Note that we now have three pairs that do not share a script: Hebrew-Norwegian, Russian-Spanish and Dutch-Arabic.

We hypothesise that parameter sharing between unrelated language pairs will be less useful in general than with related language pairs. However, it can still be useful; it has been shown previously that unrelated languages can benefit from being trained jointly. For example, Lynn et al. (2014) have shown that Indonesian was, surprisingly, particularly useful for Irish. The results are presented in Table 6.6. As expected, there is less to be gained from sharing parameters between unrelated pairs than from sharing them between related pairs: the average performance of most strategies are generally lower than they are for related languages. However, it is possible to improve over the monolingual baseline by sharing some of the parameters, positively answering **RQ4c**: *Can parameter sharing help parsing when working with unrelated languages?*

As far as **RQ4d** (*What parameters of a BiLSTM parser is it best to share for unrelated languages?*) is concerned, in general, sharing the MLP is still a helpful thing to do. It is most helpful to share the MLP and optionally one of the two other sets of parameters. We note again that it is generally better not to share characters but that this is regardless of whether or not the two languages in the pair share a script. Results are close to the monolingual baseline when everything is shared. Sharing word and character parameters but not the MLP hurts accuracy compared to the monolingual baseline.

Note finally that the setting we used in Section 6.2, hard sharing characters ✓, soft sharing words **ID** and sharing the MLP ✓performed rather well in the related languages setting (78.9 LAS on average, which is 0.7 LAS points better than the monolingual baseline and 0.2 LAS points worse than the best setting) but rather poorly in the unrelated languages setting (78.1 LAS on average which is 0.1 LAS points worse than the monolingual baseline and 1.1 LAS points worse than the best system). This can partially explain why we observed more gains when using this method with related than with unrelated languages. In addition, Ahmad et al. (2019) found that RNN architectures do not transfer well across unrelated languages compared to a Transformer architecture (Vaswani et al., 2017). It would be interesting to repeat our experiments from this section with an architecture which uses Transformers instead of LSTMs.

In this study, we explored various parameter sharing strategies and their combination for polyglot parsing. Another possibility would be to construct an architecture which learns which parameters to share. For this, we can for example use sluice networks, developed by Ruder et al. (2019). We leave that to future work.

Table 6.6. *Unrelated language pairs. Performance (LAS; in %) across select sharing strategies ranked by average performance.* MONO *is our single-task baseline;* ***W*** *refers to sharing the word parameters,* ***C*** *refers to sharing the character parameters,* ***S*** *refers to sharing the MLP parameters.*

C	W	S	he	no	fi	hr	ru	es	it	et	nl	ar	av.
MONO			80.2	80.8	70.8	76.8	82.3	83.7	83.3	70.4	77.3	76.3	78.2
LANGUAGE-BEST			80.6	81.7	72.2	77.6	82.9	84.2	84.3	72.5	78.7	76.5	79.2
✗	✗	✓	80.3	81.5	71.9	**77.6**	82.7	84.0	83.8	**72.5**	**78.7**	76.4	78.9
✗	✗	✗	80.3	**81.7**	71.9	77.7	82.5	84.0	84.2	71.8	78.5	**76.5**	78.9
✗	ID	ID	80.3	81.1	72.1	77.7	82.7	84.2	**84.2**	72.4	77.9	76.0	78.9
✗	✗	ID	79.7	81.5	72.2	77.1	82.7	83.8	84.0	72.3	78.6	76.5	78.8
ID	✗	ID	80.3	81.6	71.8	77.5	82.6	84.1	83.8	71.8	78.3	76.2	78.8
ID	✗	✗	80.0	81.6	71.5	77.2	82.7	83.9	84.0	71.1	79.0	76.4	78.7
✓	✗	ID	80.3	81.5	71.5	77.5	82.7	83.7	83.9	71.6	77.9	76.7	78.7
ID	✗	✓	80.4	81.4	71.6	77.3	82.6	83.9	84.1	71.9	77.7	76.4	78.7
✗	✓	ID	80.5	81.2	**72.2**	77.0	82.5	84.0	83.8	71.5	78.3	76.1	78.7
✗	ID	✓	**80.6**	81.1	71.9	77.1	82.7	84.2	84.0	71.9	77.1	76.2	78.7
✓	✗	✓	80.3	81.1	71.3	77.0	82.6	84.0	84.2	71.8	77.8	76.3	78.6
✗	✓	✓	80.6	81.1	71.6	76.8	82.5	83.7	83.9	71.4	78.1	76.3	78.6
✓	✗	✗	80.1	80.9	71.4	76.8	**82.9**	83.9	**84.3**	70.9	78.0	76.5	78.6
✗	ID	✗	80.3	81.3	71.2	77.4	81.9	84.2	83.9	70.7	77.6	75.8	78.4
✗	✓	✗	79.6	80.9	71.9	76.9	82.2	83.7	83.8	70.9	77.0	76.4	78.3
ID	✓	ID	80.3	81.0	70.5	76.5	82.3	83.7	83.6	71.4	77.8	76.1	78.3
ID	✓	✓	80.1	80.8	70.4	77.0	82.2	83.8	83.8	71.0	77.6	76.2	78.3
✓	✓	✓	80.5	80.9	69.8	76.6	82.3	83.7	84.0	70.6	77.4	76.2	78.2
✓	ID	ID	80.3	80.7	70.2	76.1	82.1	83.8	83.8	70.8	77.6	76.2	78.2
✓	✓	ID	79.8	80.9	71.0	76.2	82.1	83.7	83.6	70.9	77.3	76.0	78.2
ID	ID	✓	80.0	80.8	69.8	76.2	82.2	83.8	84.3	70.7	77.2	76.2	78.1
ID	ID	ID	79.8	80.5	70.1	76.6	82.1	83.9	83.8	70.6	77.2	76.3	78.1
✓	ID	✓	80.3	81.1	70.2	76.1	82.2	84.1	83.7	70.3	76.9	76.0	78.1
✓	✓	✗	80.4	80.3	70.1	76.6	82.0	83.7	83.2	69.3	76.7	76.2	77.8
✓	ID	✗	79.6	80.6	69.4	76.7	81.7	83.8	83.4	69.2	77.6	76.2	77.8
ID	✓	✗	79.6	80.1	69.9	76.6	81.8	83.5	82.9	69.2	77.6	76.3	77.7
ID	ID	✗	79.8	80.6	69.2	76.7	81.4	83.8	83.2	69.4	76.6	76.0	77.7

6.4 Conclusion

This chapter described ways in which we can leverage data from multiple treebanks to improve parsing accuracy, especially in the low-resource scenario. In Section 6.2, we have shown that we can obtain significant parsing improvements when training a polyglot parser for related languages, by making use of a treebank or language embedding to allow the network to capture language– or treebank–specific phenomena. We found this method to work well mostly when using related languages. In Section 6.3, we looked at different parameter sharing strategies when training models for related language pairs. We found that it is beneficial to share all parameters but it is most beneficial to share the MLP and sharing the word and character parameters is more or less useful depending on the language pair. We take this to mean that at the level of the configuration, the features are more abstract and less language-dependent than at the lower layers of the network. We then looked at unrelated language pairs and found that it is still useful to share the MLP but sharing the character and/or word BiLSTMs is less useful and can hurt performance if the MLP is not also shared. In this setting, it is usually best not to share too much. This can provide some explanation of why our earlier method has only been successfully applied to related languages and not to unrelated ones.

Another important contribution of this chapter is to show that we can get benefits from polyglot parsing without using gold POS tag, and without even using POS information. This answers **RQIII** positively: *Can we obtain gains from polyglot parsing without access to gold or predicted POS tags?* Other recent work by Mulcaire et al. (2019) and Vania et al. (2019) obtained further gains from polyglot parsing for low-resource languages also in a setting where POS tags are not used, further strengthening this finding.

With this, we have given some answers to **RQ4**: *How can we leverage information from multiple treebanks in different languages to improve parsing performance on the individual languages?*

7. Conclusion and Future Work

This chapter discusses our main conclusions from the thesis in Section 7.1 and avenues for future research in Section 7.2.

7.1 Conclusion

In this thesis, we have developed and analysed parsing models from a multilingual perspective. This has been made possible by the development of Universal Dependencies, a set of treebanks annotated consistently across languages. I described UD and the opportunities it opens up.

One opportunity is to build models where we can incorporate linguistic knowledge that is neither too general to be useful, nor too language-specific. I identified two aspects of language that can be incorporated into our models. The first aspect is the concept of hierarchical bias, which was argued to be a useful bias to integrate into our models from a multilingual perspective. The second aspect is the concept of dissociated nucleus. The definition of dependency trees that has been used in dependency parsing was described as a simplification compared to the tradition in dependency grammar. The concept of dissociated nucleus comes from the work of Tesnière where the basic units of syntax are not words but nuclei and where the relations that hold between words in a sentence are not all asymmetric dependency relations between a head and a dependent: some are a relation of transfer between two nodes of a dissociated nucleus, others are a relation of junction between two conjuncts.

Another opportunity created by the development of UD is to leverage data from multiple treebanks to help improve accuracy, especially in the low-resource scenario. Having consistently annotated treebanks across languages makes it easy to build models that can work for several languages, polyglot parsers.

I described the current models which are state of the art on UD treebanks. Those models make use of neural networks. With the aim to incorporate linguistic knowledge into our models, I identified three ways in which linguistic information can be used in neural NLP. The first method is to use linguistic intuition to guide architecture engineering. One example is to incorporate a hierarchical bias into our models. The second method is to use linguistic knowledge to interpret what neural networks learn. An example is to make use of diagnostic classifiers which make it possible to investigate whether vectors encode information about a task which the network was not trained for. This method can be used to test whether or not our models learn the notion of dissociated nucleus or if we have to incorporate this knowledge into the model. The last method is to use linguistic intuition to guide a multi-task learning architecture where information about different tasks are learned jointly. We see this as a special case of the first method, architecture engineering. I described work where polyglot parsing is treated as multi-task learning, where parsing different languages is considered different tasks and where some of the parameters of the model are shared and some are language-specific. This allows us to ask the question of what are the best parameter sharing strategies and if they correspond to our linguistic intuitions.

We investigated four research questions which I describe in turn. The research questions, subquestions, and their answers are summarised in Table 7.1.

RQ1 *How can we build parsing models that perform well across typologically diverse languages?*

UD treebanks open up opportunities for parsing typologically diverse languages. However, this also comes with challenges. One challenge is that UD contains a biased selection of languages. In particular, Indo-European languages are overrepresented. We proposed a set of criteria to sample a representative subset of treebanks for doing parser development and evaluation. This allows us to build models that generalise well across languages. An additional advantage is that we do not have to train parsers for a large dataset, and neural models are expensive to train and optimise. The samples for development and evaluation should ideally not be the same but I described how this is currently difficult to do with the current state of UD treebanks and we stick to one sample in this thesis.

Another challenge is dealing with morphologically complex languages. Some work has shown that modelling character sequences of words is beneficial for this type of languages, and is more robustly useful across languages than making use of POS tag information which has been very commonly used in parsing. We showed that a parser which does not use POS tags and includes character information is a strong and simple baseline and we argue that it is an appropriate model to use for hypothesis testing since it limits the number of confounding variables. Additionally, it makes the scenario more realistic for low-resource languages for which we do not necessarily have access to accurate POS taggers.

A final challenge is non-projectivity, a notorious difficulty in parsing. Non-projectivity cannot be ignored when working with a large set of treebanks because some of the treebanks have a high ratio of non-projective arcs. A solution in transition-based parsing is to add a reordering operation (swap) which increases the worst case complexity but has been shown empirically to work in expected linear time. This solution had previously not been integrated with a recent development in transition-based parser training: the use of dynamic oracles, which allow exploring errors and mitigating error propagation. In-

Table 7.1. *Summary of the research questions and answers.*

RQ0 How can linguistics inform neural NLP in the context of dependency parsing for typologically diverse languages?
RQ1 How can we build parsing models that perform well across typologically diverse languages?
RQ1a How can we develop and evaluate models with a large dataset such as UD? *Develop and evaluate on a representative sample using a set of criteria. (Section 3.1)*
RQ1b Is character information robustly useful in a BiLSTM–based parsing architecture? *Yes (Section 3.2)*
RQ1c How can we deal with non–projectivity in transition–based parsing in a convenient way? *Arc–Hybrid + reordering + static–dynamic oracle (Section 3.3)*
RQ2 How can we incorporate linguistic knowledge into neural models trained for the task of dependency parsing? *Incorporate linguistically informed inductive biases in our model architecture (Chapter 4)*
RQ2a Can a BiLSTM–based parser benefit from recursive subtree composition? *Not in a straightforward way at least (Section 4.4)*
RQ2b How does recursive composition interact with different part of the networks? *Redundancy with: the forward part of the LSTM and POS tags (Section 4.4)*
RQ3 How can we investigate what neural models learn about language when trained for the task of dependency parsing? *Analyse what our model learns using diagnostic classifiers (Chapter 5)*
RQI Does the representation of Auxiliary Verb Constructions matter in neural parsing? *Yes but less than in pre–neural parsing (Section 5.2)*
RQII Is it better to have main verbs as heads such as in UD? *Yes (Section 5.2)*
RQ3a Is information about agreement and transitivity learned by the parser? *Yes (Section 5.3)*
RQ3b Does a BiLSTM–based transition–based parser learn the notion of dissociated nucleus? *Not if main verbs are heads of AVCs such as in UD (Section 5.3)*
RQ3c Does a BiLSTM–based transition–based parser equipped with recursive subtree composition learn the notion of dissociated nucleus? *Yes (Section 5.3)*
RQ4 How can we leverage information from multiple treebanks in different languages to improve parsing performance on the individual languages? *Sharing the parameters in a polyglot parser (Chapter 6)*
RQIII Can we obtain gains from polyglot parsing without access to (gold) POS tags? *Yes, with predicted POS tags or even without POS tags at all (Chapter 6)*
RQ4a Can we improve parsing accuracy with polyglot parsing? *Yes (Section 6.2)*
RQ4b What parameters of a BiLSTM parser is it best to share for pairs of related languages? *The MLP classifier. The word and character parameters depending on the language pair (Section 6.3)*
RQ4c Can parameter sharing help when working with unrelated languages? *Yes (Section 6.3)*
RQ4b What parameters of a BiLSTM parser is it best to share for pairs of unrelated languages? *The MLP classifier. It is best not to share too many other parameters (Section 6.3)*

tegrating these two techniques is not trivial but we proposed a partial solution to it where we do not allow making errors when it comes to swap transitions. We showed empirically that this method retains the gains from using dynamic oracles and performs on par with another, more tedious method for non-projective transition-based parsing which involves pre- and post-processing (pseudo-projective parsing). It is still an open question whether we can find a full solution to integrate these two methods where we could also allow exploring errors for swap transitions.

The improvements of character modelling and non-projective parsing were integrated into our parser, UUParser, which performed competitively in the CoNLL 2017 and 2018 shared tasks, further supporting the usefulness of these methods. Equipped with a strong parser and a method for treebank sampling, we were able to better take advantages of the opportunities that UD treebanks open up as well as the opportunities that neural NLP opens up for integrating linguistic information and therefore answering the other research questions.

RQ2 *How can we incorporate linguistic knowledge into neural models for dependency parsing?*

We integrated a hierarchical bias into our model by using a recursive composition function which composes the representation of subtrees as dependents get attached to their heads. This had been done in another parser, the parser of Dyer et al. (2015), with a more complex architecture than ours but which makes use of unidirectional LSTMs only. In that parser, it has been shown to work well for two languages: English and Chinese. UD allows us to investigate this question with a diverse set of languages annotated consistently. The consistent annotation is important here: this gives us some confidence that the variance in results is due to language effects and not due to different tree representations imposed by the annotation style. We cast doubt on the finding that composition is useful for parsing: the recursive composition function seems superfluous in our parsing architecture with our treebanks sample. This indicates that BiLSTMs are powerful feature extractors that capture information about subtrees. We further analysed the interaction between different parts of the model: the recursive composition function, the forward and backward parts of the BiLSTM, part-of-speech and character information, by ablating those parts in various combinations. We found that when it comes to constructing information about the history of actions, a recursive composition function is more useful than a forward LSTM, but a forward LSTM as part of a BiLSTM is better. We found, however, that a backward LSTM with a recursive composition function is almost as good as a BiLSTM, showing that a forward LSTM and a recursive composition function are capturing similar information. We found that ablating the backward LSTM is more harmful than ablating the forward LSTM and we correlated the harmfulness of ablating a backward LSTM with language properties: right-headed languages suffer more from it than other languages. A forward LSTM and a recursive composition function perform much worse than a BiLSTM, indicating, unsurprisingly, that a backward LSTM and a recursive composition function are complementary whereas a forward LSTM and a recursive composition function are partially redundant. We concluded that sequential modelling and hierarchical modelling have advantages and disadvantages that cancel each other out in our parser when we look at parsing accuracy. Hierarchical modelling has the advantage of capturing more relevant information about subtrees but the disadvantage of suffering from error propagation: if we make an incorrect attachment, we get a representation of a subtree which does not make sense.

RQ3 *How can we investigate what neural models learn about language when trained for the task of dependency parsing?*

We next looked at whether our parsing model learns the notion of dissociated nucleus. We identified recursive composition to be potentially useful to model the relation of transfer between two nodes of a dissociated nucleus, and found that it seems to be. For this, we investigated auxiliary verb constructions in parsing. AVCs are interesting for two reasons: the choice of their representation in UD has been claimed to be suboptimal in parsing: there is some evidence in the parsing literature that it is better to have function words as heads of function word–content word relations and particularly, there is evidence that having auxiliaries as heads of AVCs is better than having the main verb as head such as is the case in UD. The second reason is that it is a typical example of a dissociated nucleus and well attested typologically. In a previous study, we re-evaluated the claim that having auxiliaries as heads of AVCs is better for parsing, and found that in the case of UD, the opposite is true. Here, we verified that this finding holds when using a neural parser. We hypothesised that the impact of using a different parsing representation for AVCs would be smaller with a neural parser than with a pre-neural parser and found that it is but that there is still a significant impact. We next looked at what our parser learns about AVCs to investigate the question of whether it learns the notion of dissociated nucleus. We proposed that if a parser learns this notion, it should learn similar information about an AVC subtree as it learns about its dissociated counterpart: finite main verbs. We used diagnostic classifiers to investigate agreement and transitivity information in AVC and finite main verb vectors. We found that our parser does not learn this notion with a UD representation of AVCs but it does if the auxiliary is the head of AVCs. We explained this by investigating the information flow in the network. We found that transitivity information is obtained by the BiLSTM whereas agreement information is encoded at the level of word type representation, either in the word type vector or in the character vector of the finite verb, and is propagated upwards through the BiLSTM but does not spill over to neighbouring words. We found that a recursive composition function allows information about agreement to transfer from the finite verb to the main verb while maintaining information about transitivity. We concluded that this allows us to learn the notion of dissociated nucleus. We take this as indication that a recursive composition function can be useful but we have not yet found the best way to integrate it in our model. We suggest that it can potentially be used to model relations of transfer, whereas it might not be necessary to model relations of dependency. We propose this as an interesting avenue for future research.

Note that having consistently annotated treebanks in multiple languages is again crucial in this study: it allows us to collect information about our construction of interest, AVCs here, while being reasonably sure that this construction is represented similarly across languages.

RQ4 *How can we leverage information from multiple treebanks in different languages to improve parsing performance on the individual languages?*

We finally looked at the benefits of treating polyglot parsing as multi-task learning. We developed a simple parameter sharing architecture where all parameters of the model are shared and where a vector representing the treebank allows the model to learn treebank-specific information. We showed that this model improves over monolingual baselines by a large margin when trained on a set of related languages. We found no clear evidence that this method works for unrelated languages.

We then investigated different parameter sharing strategies on pairs of related languages and found that sharing the parameters of the classifier of our parser is highly beneficial but the usefulness of sharing the parameters representing characters and words is more dependent on the language pair. When looking at those different parameter sharing strategies on pairs of unrelated languages, we found that sharing the classifier parameters is still a useful thing to do but sharing character and/or word parameters is less useful and can hurt parsing accuracy substantially, if the classifier is not also shared. This indicates that sharing parameters that partially abstract away from word order is useful, which makes intuitive sense. This provides some explanation for why our simple parameter sharing architecture which shares all parameters works best for related languages. We propose that we can build a hierarchical network where the classifier is shared across many languages, possibly with language family vectors allowing to learn language family specific information, and where word and character parameters are only shared across related languages. This is an interesting avenue for research. Another question we find interesting for future work is whether or not a recursive composition function might be useful in the context of multilingual parsing, since this is, like the classifier, a layer which abstracts away from word order.

We also found that we can get benefits from polyglot parsing without using gold POS tags, which a lot of previous work on the topic were using, and even without using POS tags at all.

At a more general level, the overarching question driving this thesis was the following:

RQ0 How can linguistics inform neural NLP?

We looked at one way of incorporating linguistic knowledge into neural NLP models: through architecture engineering. We gave the example of incorporating a hierarchical bias for dependency parsing and showed how its effect can be studied systematically, by investigating its interaction with different parts of the network. We showed how to test whether a neural network learns a specific linguistic phenomenon: the concept of dissociated nucleus. We can investigate different phenomena in a similar way. At the same time, we showed a way in which we can characterise what neural networks learn about language by looking at agreement and transitivity information at different layers in our network. We finally investigated polyglot parsing and found that the classifier which scores transitions is a layer that is almost always useful to share, regardless of whether the languages involved are related or unrelated. This makes linguistic sense since that layer is the most abstract and partially abstracts away from word order. We found that sharing the character and word layers is differently useful depending on the language pairs. This makes sense linguistically but also due to the architecture used: LSTMs are good at learning sequential patterns such as word order. These types of neural networks are therefore more suitable to share among related languages and maybe they are not the best to use if we want to build parsers that share character and word parameters for unrelated languages. We have therefore shown various ways in which linguistics can inform neural NLP in the context of dependency parsing and these can be adapted to other NLP areas.

I introduced this thesis by saying that multilingual resources for NLP are increasing which leads to an increase in studies on cross-lingual learning. UD offers a great scenario for building general models for dependency parsing: it covers a large range of typologically diverse languages and uses a consistent annotation scheme. Such cross-linguistically consistent multilingual resources are still scarce for other tasks but are increasing. For example, UniMorph (Sylak-Glassman, 2016) is a large collection of morphological dictionaries. We have suggested ways in which we can take advantage of such a resource, if it exists, to make sure that we build models that work for typologically diverse languages. We have argued that we should sample subsets of the dataset for development and evaluation to be representative of world languages. We also showed how to work in a polymonolingual setting which makes it possible to build models that generalise well across languages but also allows us to find out how different types of architectures correlate with language properties. For example, we saw how character models are more useful for more morphologically complex languages. We showed how we can use polyglot training when we have a model that we know works well for typologically diverse languages.

In summary, this work has led to the following contributions:

Contributions

- Extension of a parser now close to state of the art for parsing typologically diverse languages while being simple.
- Extension of a method for conveniently dealing with non-projectivity in transition-based parsing while keeping some of the benefits of dynamic oracles.
- A general method for working with a multilingual dataset which is annotated with a universal scheme.
- A study on the impact of using a tree layer in parsing typologically varied languages. This includes a detailed analysis of the interaction of the tree layer with other parts of the model.
- A study on the impact of representation choice of auxiliary verb constructions on parsing accuracy in neural parsing.
- Methods and results to assess linguistic plausibility in neural parsing. This includes an extensive study of agreement and transitivity information learned by a parser for auxiliary verb constructions.
- Developments and analysis in polyglot parsing for related and unrelated languages.

7.2 Future Work

In this thesis, I have described 1) ways to integrate linguistic knowledge into parsing models in a polymonolingual setting and 2) ways in which we can leverage data from multiple treebanks in a polyglot parsing setting. Ultimately, an obvious goal of multilingual parsing is to build a polyglot model which is capable of parsing many languages, and incorporating linguistic knowledge into those models can be useful. In other words, we can combine these two approaches. A way to combine these two lines of research would be to first test how inductive biases affect different languages. We can then find the ones that are universally useful across languages and the ones that are useful for specific language types. We can subsequently build polyglot models that incorporate the universal inductive biases as well as the typological knowledge needed to deal with language specific phenomena.

We have not exploited the incorporation of typological knowledge into our models in this thesis but this is an active area of research which has still a lot to offer (see O'Horan et al. (2016) for a survey of the use of typological information in NLP). Work on incorporating typological knowledge into parsers has shown some promise with pre-neural parsers (Naseem et al., 2012; Täckström et al., 2013; Zhang and Barzilay, 2015). Such studies with neural parsers have shown mixed results. Ammar et al. (2016) found no benefit from using typologically informed language representations in their polyglot parser, Scholivet et al. (2019) and Fisch et al. (2019) found mixed results from using typological features in delexicalized cross-lingual parsing. Additionally, all of these studies have used gold POS tags as part of the input representation, except for Ammar et al. (2016) who found large decreases in accuracy from using predicted POS tags compared to gold ones. It remains to be seen whether typological information can be useful in (neural) polyglot parsing that uses either predicted POS tags or no POS tags at all.

A limitation of work in this thesis is the use of a single base architecture: a BiLSTM-based parser. As mentioned in Chapter 6, Transformers might be more appropriate for polyglot parsing, because they do not rely on word order. This is an interesting future direction. It would also be interesting to investigate the impact of using recursive composition with a Transformer-based parser, as well as compare what BiLSTMs and Transformers learn about AVCs and other types of dissociated nuclei.

Another limitation is the use of a transition-based algorithm throughout. It would be interesting to see if graph-based parsers learn different things about AVCs and dissociated nuclei in general. Polyglot parsing might also work better with a graph-based than with a transition-based parser. This is because the parsing algorithm completely abstracts away from word order. While a transition-based parser differentiates attaching dependency relations via a right- or left-arc transition, a graph-based parser scores dependency relations between any two words in a sentence, regardless of their position in the sentence.

The focus on AVCs allowed in-depth analysis, but it would also be interesting to expand our work to other types of dissociated nucleus.

In this thesis, I asked the question of how linguistics can inform neural NLP. The converse question is also interesting: how can neural NLP inform linguistics? Finding what a neural model learns about language can be interesting from a linguistic perspective, as well as testing the incorporation of linguistically informed biases into our models. Linzen (2018) discusses how the incorporation of a hierarchical bias into neural models can inform language acquisition. He argues that if such a hierarchical bias is crucial to perform well on a task, this may indicate that this is a crucial property in language acquisition. Linzen (2018) discusses more generally how linguistics and deep learning can contribute to each other, highlighting the potential of doing so.

Overall, the development of UD and advances in neural dependency parsing have opened up many opportunities and this thesis has shown ways in which we can take advantage of this as well as taken several steps in this direction.

References

Yossi Adi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg. 2017. Fine-grained analysis of sentence embeddings using auxiliary prediction tasks. In *International Conference on Learning Representations*.

Željko Agić, Anders Johannsen, Barbara Plank, Héctor Martínez Alonso, Natalie Schluter, and Anders Søgaard. 2016. Multilingual projection for parsing truly low-resource languages. *Transactions of the Association for Computational Linguistics*, 4:301–312.

Wasi Ahmad, Zhisong Zhang, Xuezhe Ma, Eduard Hovy, Kai-Wei Chang, and Nanyun Peng. 2019. On Difficulties of Cross-Lingual Transfer with Order Differences: A Case Study on 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 2440–2452.

Waleed Ammar, George Mulcaire, Miguel Ballesteros, Chris Dyer, and Noah Smith. 2016. Many Languages, One Parser. *Transactions of the Association for Computational Linguistics*, 4:431–444.

Gregory DS Anderson. 2011. Auxiliary Verb Constructions (and Other Complex Predicate Types): A Functional–Constructional Overview. *Language and Linguistics Compass*, 5(11):795–828.

Giuseppe Attardi. 2006. Experiments with a Multilanguage Non-Projective Dependency Parser. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 166–170.

Giuseppe Attardi, Felice Dell'Orletta, Maria Simi, and Joseph Turian. 2009. Accurate dependency parsing with a stacked multilayer perceptron. In *Proceedings of EVALITA 9*, pages 1–8.

Giuseppe Attardi, Simone Saletti, and Maria Simi. 2015. Evolution of Italian Treebank and Dependency Parsing towards Universal Dependencies. In *Proceedings of the Second Italian Conference on Computational Linguistics CLiC-it 2015*, pages 25–30.

Lauriane Aufrant. 2018. *Training parsers for low-resourced languages: improving cross-lingual transfer with monolingual knowledge*. Ph.D. thesis, Paris Saclay.

Miguel Ballesteros, Chris Dyer, and Noah A Smith. 2015. Improved Transition-based Parsing by Modeling Characters instead of Words with LSTMs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 349–359.

Ali Basirat and Joakim Nivre. 2017. Real-valued Syntactic Word Vectors (RSV) for Greedy Neural Dependency Parsing. In *Proceedings of the 21st Nordic Conference on Computational Linguistics (NoDaLiDa)*, pages 21–28.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam

Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.

Yonatan Belinkov, Nadir Durrani, Fahim Dalvi, Hassan Sajjad, and James Glass. 2017. What do Neural Machine Translation Models Learn about Morphology? In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 861–872.

Yonatan Belinkov and James Glass. 2017. Analyzing Hidden Representations in End-to-End Automatic Speech Recognition Systems. In I. Guyon, U. V Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2441–2451. Curran Associates, Inc.

Yonatan Belinkov and James Glass. 2019. Analysis methods in neural language processing: A survey. *Transactions of the Association for Computational Linguistics*, 7:49–72.

Emily M Bender. 2011. On achieving and evaluating language-independence in NLP. *Linguistic Issues in Language Technology*, 6(3):1–26.

Aleksandrs Berdicevskis, Çağrı Çöltekin, Katharina Ehret, Kilu von Prince, Daniel Ross, Bill Thompson, Chunxiao Yan, Vera Demberg, Gary Lupyan, Taraka Rama, and Christian Bentz. 2018. Using universal dependencies in cross-linguistic complexity research. In *Proceedings of the Second Workshop on Universal Dependencies (UDW 2018)*, pages 8–17.

Anders Björkelund and Joakim Nivre. 2015. Non-Deterministic Oracles for Unrestricted Non-Projective Transition-Based Dependency Parsing. In *Proceedings of the 14th International Conference on Parsing Technologies*, pages 76–86.

Bernd Bohnet, Ryan McDonald, Gonçalo Simões, Daniel Andor, Emily Pitler, and Joshua Maynez. 2018. Morphosyntactic tagging with a meta-BiLSTM model over context sensitive token encodings. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2642–2652.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Lingustics*, 5:135–146.

Cristina Bosco, Simonetta Montemagni, Alessandro Mazzei, Vincenzo Lombardo, Felice Dell'Orletta, and Alessandro Lenci. 2009. Evalita'09 Parsing Task: comparing dependency parsers and treebanks. *Proceedings of EVALITA*, 9.

Matthias Buch-Kromann. 2003. The Danish dependency treebank and the dtag treebank tool. In *2nd Workshop on Treebanks and Linguistic Theories (TLT), Sweden*, pages 217–220.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL)*, pages 149–164.

Xavier Carreras, Mihai Surdeanu, and Lluis Marquez. 2006. Projective dependency parsing with perceptron. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 181–185.

Wanxiang Che, Jiang Guo, Yuxuan Wang, Bo Zheng, Huaipeng Zhao, Yang Liu, Dechuan Teng, and Ting Liu. 2017. The HIT-SCIR system for end-to-end parsing of universal dependencies. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 52–62.

Wanxiang Che, Yijia Liu, Yuxuan Wang, Bo Zheng, and Ting Liu. 2018. Towards Better UD Parsing: Deep Contextualized Word Embeddings, Ensemble, and Treebank Concatenation. *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 55–64.

Danqi Chen and Christopher Manning. 2014. A Fast and Accurate Dependency Parser using Neural Networks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750.

Kehai Chen, Rui Wang, Masao Utiyama, Lemao Liu, Akihiro Tamura, Eiichiro Sumita, and Tiejun Zhao. 2017. Neural machine translation with source dependency representation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2846–2852.

Noam Chomsky. 1957. *Syntactic Structures*. Mouton.

Yoeng-Jin Chu and Tseng-hong Liu. 1965. On the shortest arborescence of a directed graph. *Scientia Sinica*, 14:1396–1400.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493–2537.

William Croft, Dawn Nordquist, Katherine Looney, and Michael Regan. 2017. Linguistic Typology meets Universal Dependencies. In *Proceedings of the 15th Treebanks and Linguistic Theories Workshop (TLT)*, pages 63–75.

Fahim Dalvi, Nadir Durrani, Hassan Sajjad, Yonatan Belinkov, and Stephan Vogel. 2017. Understanding and Improving Morphological Learning in the Neural Machine Translation Decoder. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 142–151.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Robert MW Dixon and Alexandra Y Aikhenvald. 2000. *Changing valency: Case studies in transitivity*, volume 413. Cambridge University Press.

Timothy Dozat and Christopher Manning. 2017. Deep Biaffine Attention for Neural Dependency Parsing. In *Proceedings of the 5th International Conference on Learning Representations.*

Timothy Dozat, Peng Qi, and Christopher D Manning. 2017. Stanford's Graph-based Neural Dependency Parser at the CoNLL 2017 Shared Task. *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 20–30.

Matthew S Dryer and Martin Haspelmath, editors. 2013. *WALS Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig.

Chris Dyer. 2017. Should Neural Network Architecture Reflect Linguistic Structure? In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, page 1.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A Smith. 2015. Transition-Based Dependency Parsing with Stack Long Short-Term Memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 334–343.

Chris Dyer, Adhiguna Kuncoro, Miguel Ballesteros, and Noah A. Smith. 2016. Recurrent neural network grammars. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 199–209.

Jack Edmonds. 1967. Optimum Branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240.

Jason M Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING)*, pages 340–345.

Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science*, 14(2):179–211.

Émile Enguehard, Yoav Goldberg, and Tal Linzen. 2017. Exploring the Syntactic Abilities of RNNs with Multi-task Learning. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 3–14.

Allyson Ettinger, Ahmed Elgohary, and Philip Resnik. 2016. Probing for semantic evidence of composition by means of simple classification tasks. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*, pages 134–139.

Martin BH Everaert, Marinus AC Huybregts, Noam Chomsky, Robert C Berwick, and Johan J Bolhuis. 2015. Structures, not strings: linguistics as part of the cognitive sciences. *Trends in cognitive sciences*, 19(12):729–743.

Daniel L Everett. 2005. Cultural Constraints on Grammar and Cognition in Piraha. *Current Anthropology*, 46(4).

Agnieszka Falenska and Jonas Kuhn. 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 (ACL)*, pages 117–128.

Adam Fisch, Jiang Guo, and Regina Barzilay. 2019. Working hard or hardly working: Challenges of integrating typology into neural dependency parsers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Timothy Q Gentner, Kimberly M Fenn, Daniel Margoliash, and Howard C Nusbaum. 2006. Recursive syntactic pattern learning by songbirds. *Nature*, 440(7088):1204.

Yoav Goldberg. 2017. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309.

Yoav Goldberg and Joakim Nivre. 2012. A Dynamic Oracle for Arc-Eager Dependency Parsing. In *Proceedings of the 24th International Conference on Computational Linguistics (COLING)*, pages 959–976.

Yoav Goldberg and Joakim Nivre. 2013. Training Deterministic Parsers with Non-Deterministic Oracles. *Transactions of the Association for Computational Linguistics*, 1:403–414.

Carlos Gómez-Rodríguez and Daniel Fernández-González. 2015. An efficient dynamic oracle for unrestricted non-projective 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 2: Short Papers)*, pages 256–261.

Carlos Gómez-Rodríguez and Joakim Nivre. 2010. A Transition-Based Parser for 2-Planar Dependency Structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1492–1501.

Johannes Gontrum. 2019. Attention Mechanisms for Transition-based Dependency Parsing. Master's thesis, Uppsala University.

Kristina Gulordava, Piotr Bojanowski, Edouard Grave, Tal Linzen, and Marco Baroni. 2018. Colorless green recurrent networks dream hierarchically. 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 1195–1205.

Kristina Gulordava and Paola Merlo. 2016. Multi-lingual dependency parsing evaluation: a large-scale analysis of word order properties using artificial data. *Transactions of the Association for Computational Linguistics*, 4:343–356.

Jiang Guo, Wanxiang Che, Haifeng Wang, and Ting Liu. 2016. A universal framework for inductive transfer parsing across multi-typed treebanks. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 12–22.

Joakim Gylling. 2017. Transition-Based Dependency Parsing with Neural Networks. Master's thesis, Linköping University.

Jan Hajic, Eva Hajicova, Petr Pajas, Jarmila Panevova, and Petr Sgall. 2001. Prague Dependency Treebank 1.0. LDC, 2001T10.

Eva Hajičová. 2011. Computational linguistics without linguistics? A view from prague. *Linguistic Issues in Language Technology*, 6.

Johan Hall, Jens Nilsson, and Joakim Nivre. 2010. Single malt or blended? a study in multilingual parser optimization. In *Trends in Parsing Technology*, pages 19–33. Springer.

Hanna-Ilona Härmävaara. 2014. Facilitating mutual understanding in everyday interaction between finns and estonians. *Applied Linguistics Review*, 5(1):211–245.

Marc D Hauser, Noam Chomsky, and W Tecumseh Fitch. 2002. The Faculty of Language: What Is It, Who Has It, and How Did It Evolve? *Science*, 298(5598):1569–1579.

John Hewitt and Christopher D Manning. 2019. A Structural Probe for Finding Syntax in Word Representations. 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 4129–4138.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Liang Huang, Wenbin Jiang, and Qun Liu. 2009. Bilingually-constrained (monolingual) shift-reduce parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3-Volume 3*, pages 1222–1231.

Dieuwke Hupkes, Sara Veldhoen, and Willem Zuidema. 2018. Visualisation and ‘Diagnostic Classifiers’ Reveal how Recurrent and Recursive Neural Networks Process Hierarchical Structure. *Journal of Artificial Intelligence Research*, 61:907–926.

Richard Johansson. 2013. Training parsers on incompatible treebanks. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 127–137.

Mark Johnson et al. 2011. How relevant is linguistics to computational linguistics. *Linguistic Issues in Language Technology*, 6(7).

Eliyahu Kiperwasser and Yoav Goldberg. 2016a. Easy-first dependency parsing with hierarchical tree LSTMs. *Transactions of the Association for Computational Linguistics*, 4:445–461.

Eliyahu Kiperwasser and Yoav Goldberg. 2016b. Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL, Demo and Poster Sessions*, pages 177–180.

Lingpeng Kong. 2017. *Neural Representation Learning in Linguistic Structured Prediction*. Ph.D. thesis, Carnegie Mellon University.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proceedings of ACL-08: HLT*, pages 595–603.

Sandra Kübler, Ryan McDonald, and Joakim Nivre. 2009. *Dependency Parsing*. Morgan and Claypool.

Marco Kuhlmann, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Dynamic Programming Algorithms for Transition-Based Dependency Parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 673–682.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, Graham Neubig, and Noah A Smith. 2017. What Do Recurrent Neural Network Grammars Learn About Syntax? In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1249–1258.

Adhiguna Kuncoro, Chris Dyer, John Hale, Dani Yogatama, Stephen Clark, and Phil Blunsom. 2018. LSTMs can learn syntax-sensitive dependencies well, but modeling structure makes them better. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1426–1436.

Guillaume Lample, Alexis Conneau, Marc’Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. 2018. Word translation without parallel data. In *International Conference on Learning Representations*.

Minh Lê and Antske Fokkens. 2017. Tackling error propagation through reinforcement learning: A case of greedy dependency parsing. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational*

Linguistics: Volume 1, Long Papers, pages 677–687.

Gaël Le Godais, Tal Linzen, and Emmanuel Dupoux. 2017. Comparing Character-level Neural Language Models Using a Lexical Decision Task. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 125–130.

Miryam de Lhoneux, Miguel Ballesteros, and Joakim Nivre. 2019a. 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.

Miryam de Lhoneux, Johannes Bjerva, Isabelle Augenstein, and Anders Søgaard. 2018. Parameter sharing between dependency parsers for related languages. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4992–4997.

Miryam de Lhoneux and Joakim Nivre. 2016. Should Have, Would Have, Could Have. Investigating Verb Group Representations for Parsing with Universal Dependencies. In *Proceedings of the Workshop on Multilingual and Cross-lingual Methods in NLP*, pages 10–19.

Miryam de Lhoneux, Yan Shao, Ali Basirat, Eliyahu Kiperwasser, Sara Stymne, Yoav Goldberg, and Joakim Nivre. 2017a. 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.

Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2017b. Arc-Hybrid Non-Projective Dependency Parsing with a Static-Dynamic Oracle. In *Proceedings of the 15th International Conference on Parsing Technologies*, pages 99–104.

Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2017c. Old School vs. New School: Comparing Transition-Based Parsers with and without Neural Network Enhancement. In *Proceedings of the 15th Treebanks and Linguistic Theories Workshop (TLT)*, pages 99–110.

Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2019b. What Should/Do/Can LSTMs Learn When Parsing Auxiliary Verb Constructions? *arXiv preprint arXiv:1907.07950*. Under review.

Tal Linzen. 2018. What can linguistics and deep learning contribute to each other? *arXiv preprint arXiv:1809.04179*.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of LSTMs to learn syntax-sensitive dependencies. *Transactions of the Association for Computational Linguistics*, 4:521–535.

Teresa Lynn, Jennifer Foster, Mark Dras, and Lamia Tounsi. 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.

Xuezhe Ma, Zhengzhong Liu, and Eduard Hovy. 2016. Unsupervised ranking model for entity coreference resolution. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1012–1018.

Christopher D Manning. 2015. Computational linguistics and deep learning. *Computational Linguistics*, 41(4):701–707.

Marie-Catherine de Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D Manning. 2014. Universal Stanford Dependencies: A cross-linguistic typology. In *Proceedings of the 9th International Conference on Language Resources and Evaluation (LREC)*, pages 4585–4592.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, pages 449–454.

Marie-Catherine de Marneffe and Joakim Nivre. 2019. Dependency Grammar. *Annual Review of Linguistics*, 5:197–218.

Rebecca Marvin and Tal Linzen. 2018. Targeted Syntactic Evaluation of Language Models. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1192–1202.

David McClosky, Eugene Charniak, and Mark Johnson. 2010. Automatic domain adaptation for parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 28–36.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Spanning Tree Methods for Discriminative Training of Dependency Parsers. Technical Report MS-CIS-05-11, University of Pennsylvania, Department of Computer and Information Science.

Ryan McDonald and Joakim Nivre. 2007. Characterizing the Errors of Data-Driven Dependency Parsing Models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 122–131.

Ryan McDonald and Joakim Nivre. 2011. Analyzing and Integrating Dependency Parsers. *Computational Linguistics*, pages 197–230.

Ryan McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith Hall, Slav Petrov, Hao Zhang, Oscar Täckström, Claudia Bedini, Núria Bertomeu Castelló, and Jungmee Lee. 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.

Ryan McDonald, Slav Petrov, and Keith Hall. 2011. Multi-Source Transfer of Delexicalized Dependency Parsers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 62–72.

Igor Mel'čuk. 1988. *Dependency Syntax: Theory and Practice*. State University of New York Press.

Paola Merlo. 2019. Probing word and sentence embeddings for long-distance dependencies effects in French and English. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 158–172.

Paola Merlo and Francesco Ackermann. 2018. Vectorial semantic spaces do not encode human judgments of intervention similarity. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 392–401.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint* *arXiv:1301.3781*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013c. Distributed Representations of Words and Phrases and their Compositionality. In C. J C Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

Phoebe Mulcaire, Jungo Kasai, and Noah A Smith. 2019. Low-resource parsing with crosslingual contextualized representations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Tahira Naseem, Regina Barzilay, and Amir Globerson. 2012. Selective sharing for multilingual dependency parsing. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 629–637.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, et al. 2017. DyNet: The Dynamic Neural Network Toolkit. *arXiv preprint arXiv:1701.03980*.

Jens Nilsson and Joakim Nivre. 2008. MaltEval: An Evaluation and Visualization Tool for Dependency Parsing. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC)*, pages 161–166.

Jens Nilsson, Joakim Nivre, and Johan Hall. 2006. Graph Transformations in Data-Driven Dependency Parsing. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, pages 257–264.

Jens Nilsson, Joakim Nivre, and Johan Hall. 2007. Generalizing Tree Transformations for Inductive Dependency Parsing. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 968–975.

Joakim Nivre. 2003. An Efficient Algorithm for Projective Dependency Parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.

Joakim Nivre. 2004. Incrementality in Deterministic Dependency Parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together (ACL)*, pages 50–57.

Joakim Nivre. 2006. *Inductive Dependency Parsing*. Springer.

Joakim Nivre. 2007. Incremental Non-Projective Dependency Parsing. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT)*, pages 396–403.

Joakim Nivre. 2009. Non-Projective Dependency Parsing in Expected Linear Time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP (ACL-IJCNLP)*, pages 351–359.

Joakim Nivre. 2015. Towards a Universal Grammar for Natural Language Processing. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, pages 3–16. Springer.

Joakim Nivre, Mitchell Abrams, Željko Agić, Lars Ahrenberg, Gabrielė Aleksandravičiūtė, Lene Antonsen, Katya Aplonova, Maria Jesus Aranzabe, Gashaw Arutie, Masayuki Asahara, Luma Ateyah, Mohammed Attia, Aitziber Atutxa, Liesbeth Augustinus, Elena Badmaeva, Miguel Ballesteros, Esha Banerjee, Sebastian Bank, Verginica Barbu Mititelu, Victoria Basmov, John Bauer, Sandra Bellato, Kepa Bengoetxea, Yevgeni Berzak, Irshad Ahmad Bhat, Riyaz Ahmad Bhat, Erica Biagetti, Eckhard Bick, Agnė Bielinskienė, Rogier Blokland, Victoria Bobicev, Loïc Boizou, Emanuel Borges Völker, Carl Börstell, Cristina Bosco, Gosse Bouma, Sam Bowman, Adriane Boyd, Kristina Brokaitė, Aljoscha Burchardt, Marie Candito, Bernard Caron, Gauthier Caron, Gülşen Cebiroğlu Eryiğit, Flavio Massimiliano Cecchini, Giuseppe G. A. Celano, Slavomír Čéplö, Savas Cetin, Fabricio Chalub, Jinho Choi, Yongseok Cho, Jayeol Chun, Silvie Cinková, Aurélie Collomb, Çağrı Çöltekin, Miriam Connor, Marine Courtin, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraza, Carly Dickerson, Bamba Dione, Peter Dirix, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Hanne Eckhoff, Marhaba Eli, Ali Elkahky, Binyam Ephrem, Tomaž Erjavec, Aline Etienne, Richárd Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Kazunori Fujita, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Moa Gärdenfors, Sebastian Garza, Kim Gerdes, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta González Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Céline Guillot-Barbance, Nizar Habash, Jan Hajič, Jan Hajič jr., Linh Hà Mỹ, Na-Rae Han, Kim Harris, Dag Haug, Johannes Heinecke, Felix Hennig, Barbora Hladká, Jaroslava Hlaváčová, Florinel Hociung, Petter Hohle, Jena Hwang, Takumi Ikeda, Radu Ion, Elena Irimia, Ọlájídé Ishola, Tomáš Jelínek, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşıkara, Andre Kaasen, Sylvain Kahane, Hiroshi Kanayama, Jenna Kanerva, Boris Katz, Tolga Kayadelen, Jessica Kenney, Václava Kettnerová, Jesse Kirchner, Arne Köhn, Kamil Kopacewicz, Natalia Kotsyba, Jolanta Kovalevskaitė, Simon Krek, Sookyoung Kwak, Veronika Laippala, Lorenzo Lambertino, Lucia Lam, Tatiana Lando, Septina Dian Larasati, Alexei Lavrentiev, John Lee, Phương Lê Hồng, Alessandro Lenci, Saran Lertpradit, Herman Leung, Cheuk Ying Li, Josie Li, Keying Li, KyungTae Lim, Yuan Li, Nikola Ljubešić, Olga Loginova, Olga Lyashevskaya, Teresa Lynn, Vivien Macketanz, Aibek Makazhanov, Michael Mandl, Christopher Manning, Ruli Manurung, Cătălina Mărănduc, David Mareček, Katrin Marheinecke, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Sarah McGuinness, Gustavo Mendonça, Niko Miekka, Margarita Misirpashayeva, Anna Missilä, Cătălin Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Keiko Sophie Mori, Tomohiko Morioka, Shinsuke Mori, Shigeki Moro, Bjartur Mortensen, Bohdan Moskalevskyi, Kadri Muischnek, Yugo Murawaki, Kaili Müürisep, Pinkey Nainwani, Juan Ignacio Navarro Horñiacek, Anna Nedoluzhko, Gunta Nešpore-Bērzkalne, Lương Nguyễn Thị, Huyền Nguyễn Thị Minh, Yoshihiro Nikaido, Vitaly Nikolaev, Rattima Nitisaroj, Hanna Nurmi, Stina Ojala, Adédayọ̀ Olúòkun, Mai Omura, Petya Osenova, Robert Östling, Lilja Øvrelid, Niko Partanen, Elena Pascual, Marco Passarotti, Agnieszka Patejuk, Guilherme Paulino-Passos, Angelika Peljak-Łapińska, Siyao Peng, Cenel-Augusto Perez, Guy Perrier, Daria Petrova, Slav Petrov, Jussi Piitulainen, Tommi A Pirinen, Emily Pitler, Barbara Plank, Thierry Poibeau, Martin Popel, Lauma Pretkalniņa, Sophie Prévost, Prokopis Prokopidis, Adam Przepiórkowski, Tiina Puolakainen, Sampo Pyysalo, Andriela Rääbis, Alexandre Rademaker, Loganathan Ramasamy, Taraka Rama, Carlos Ramisch, Vinit Ravishankar, Livy Real, Siva Reddy, Georg Rehm, Michael Rießler, Erika Rimkutė, Larissa Rinaldi, Laura Rituma, Luisa Rocha, Mykhailo Romanenko, Rudolf Rosa, Davide Rovati, Valentin Roșca, Olga Rudina, Jack Rueter, Shoval Sadde, Benoît Sagot, Shadi Saleh, Alessio Salomoni, Tanja Samardžić, Stephanie Samson, Manuela Sanguinetti, Dage Särg, Baiba Saulīte, Yanin Sawanakunanon, Nathan Schneider, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Mo Shen, Atsuko Shimada, Hiroyuki Shirasu, Muh Shohibussirri, Dmitry Sichinava, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron Smith, Isabela Soares-Bastos, Carolyn Spadine, Antonio Stella, Milan Straka, Jana Strnadová, Alane Suhr, Umut Sulubacak, Shingo Suzuki, Zsolt Szántó, Dima Taji, Yuta Takahashi, Fabio Tamburini, Takaaki Tanaka, Isabelle Tellier, Guillaume Thomas, Liisi Torga, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Zdeňka Urešová, Larraitz Uria, Hans Uszkoreit, Sowmya Vajjala, Daniel van Niekerk, Gertjan van Noord, Viktor Varga, Eric Villemonte de la Clergerie, Veronika Vincze, Lars Wallin, Abigail Walsh, Jing Xian Wang, Jonathan North Washington, Maximilan Wendt, Seyi Williams, Mats Wirén, Christian Wittern, Tsegay Woldemariam, Tak-sum Wong, Alina Wróblewska, Mary Yako, Naoki Yamazaki, Chunxiao Yan, Koichi Yasuoka, Marat M. Yavrumyan, Zhuoran Yu, Zdeněk Žabokrtský, Amir Zeldes, Daniel Zeman, Manying Zhang, and Hanzhi Zhu. 2019. Universal dependencies 2.4. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Joakim Nivre and Chiao-Ting Fang. 2017. Universal dependency evaluation. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies, 22 May, Gothenburg Sweden*, 135, pages 86–95.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 915–932.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2006. MaltParser: A Data-Driven Parser-Generator for Dependency Parsing. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC)*, pages 2216–2219.

Joakim Nivre, Marco Kuhlmann, and Johan Hall. 2009. An Improved Oracle for Dependency Parsing with Online Reordering. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 73–76.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016a. Universal Dependencies v1: A multilingual treebank collection. In *Proceedings of the 10th International*

Conference on Language Resources and Evaluation (LREC 2016), pages 1659–1666.

Joakim Nivre and Beata Megyesi. 2007. Bootstrapping a Swedish Treebank using cross-corpus harmonization and annotation projection. In *Proceedings of the 6th International Workshop on Treebanks and Linguistic Theories*, pages 97–102.

Joakim Nivre and Jens Nilsson. 2005. Pseudo-Projective Dependency Parsing. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 99–106.

Joakim Nivre et al. 2015. Universal Dependencies 1.2. LINDAT/CLARIN digital library at the Institute of - Formal and Applied Linguistics, Charles University, - Prague.

Joakim Nivre et al. 2016b. Universal Dependencies 1.3. LINDAT/CLARIN digital library at the Institute of - Formal and Applied Linguistics, Charles University, - Prague.

Joakim Nivre et al. 2017a. Universal Dependencies 2.0. LINDAT/CLARIN digital library at the Institute of - Formal and Applied Linguistics, Charles University, - Prague.

Joakim Nivre et al. 2017b. Universal Dependencies 2.1. LINDAT/CLARIN digital library at the Institute of - Formal and Applied Linguistics, Charles University, - Prague.

Joakim Nivre et al. 2018. Universal Dependencies 2.2. LINDAT/CLARIN digital library at the Institute of - Formal and Applied Linguistics, Charles University, - Prague.

Helen O'Horan, Yevgeni Berzak, Ivan Vulić, Roi Reichart, and Anna Korhonen. 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.

Timothy Osborne and Daniel Maxwell. 2015. A historical overview of the status of function words in dependency grammar. In *Proceedings of the Third International Conference on Dependency Linguistics (Depling 2015)*, pages 241–250.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237.

Slav Petrov, Dipanjan Das, and Ryan McDonald. 2012. A universal part-of-speech tagset. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*, pages 2089–2096.

Barbara Plank. 2019. Transferring NLP models across languages and domains. Invited talk at SyntaxFest.

Geoffrey K Pullum. 2009. Computational linguistics and generative linguistics: The triumph of hope over experience. In *Proceedings of the EACL 2009 Workshop on the Interaction between Linguistics and Computational Linguistics: Virtuous, Vicious or Vacuous?*, pages 12–21.

Peng Qi, Timothy Dozat, Yuhao Zhang, and Christopher D. Manning. 2018. Universal dependency parsing from scratch. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 160–170.

Shauli Ravfogel, Yoav Goldberg, and Tal Linzen. 2019. Studying the inductive biases of RNNs with synthetic variations of natural languages. 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 3532–3542.

Shauli Ravfogel, Yoav Goldberg, and Francis Tyers. 2018. Can LSTM Learn to Capture Agreement? The Case of Basque. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 98–107.

Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50.

Tanya Reinhart. 1981. Definite NP anaphora and c-command domains. *Linguistic Inquiry*, 12(4):605–635.

Rudolf Rosa. 2015. Multi-source cross-lingual delexicalized parser transfer: Prague or stanford? In *Proceedings of the Third International Conference on Dependency Linguistics (Depling 2015)*, pages 281–290.

Sebastian Ruder. 2016. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Sebastian Ruder. 2019. *Neural Transfer Learning for Natural Language Processing*. Ph.D. thesis, National University of Ireland, Galway.

Sebastian Ruder, Joachim Bingel, Isabelle Augenstein, and Anders Søgaard. 2019. Latent multi-task architecture learning. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*.

Kenji Sagae and Alon Lavie. 2006. Parser Combination by Reparsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 129–132.

Manon Scholivet, Franck Dary, Alexis Nasr, Benoit Favre, and Carlos Ramisch. 2019. Typological features for multilingual delexicalised 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 3919–3930.

Klaus Schubert. 1987. *Metataxis: Contrastive dependency syntax for machine translation*, volume 2. Foris Publications.

Roy Schwartz, Omri Abend, and Ari Rappoport. 2012. Learnability-Based Syntactic Annotation Design. In *Proceedings of the 24th International Conference on Computational Linguistics (COLING)*, pages 2405–2422.

Petr Sgall, Eva Hajičová, and Jarmila Panevová. 1986. *The Meaning of the Sentence in Its Pragmatic Aspects*. Reidel.

Yan Shao, Christian Hardmeier, and Joakim Nivre. 2018. Universal Word Segmentation: Implementation and Interpretation. *Transactions of the Association for Computational Linguistics*, 6:421–435.

Yan Shao, Christian Hardmeier, Jörg Tiedemann, and Joakim Nivre. 2017. Character-Based Joint Segmentation and POS Tagging for Chinese using Bidirectional RNN-CRF. In *The 8th International Joint Conference on Natural Language Processing*, pages 173–183.

Xing Shi, Inkit Padhi, and Kevin Knight. 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.

Natalia Silveira and Christopher Manning. 2015. Does universal dependencies need a parsing representation? an investigation of english. In *Proceedings of the Third International Conference on Dependency Linguistics (Depling 2015)*, pages 310–319.

Aaron Smith, Bernd Bohnet, Miryam de Lhoneux, Joakim Nivre, Yan Shao, and Sara Stymne. 2018a. 82 treebanks, 34 models: Universal dependency parsing with multi-treebank models. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 113–123.

Aaron Smith, Miryam de Lhoneux, Sara Stymne, and Joakim Nivre. 2018b. 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.

Richard Socher, John Bauer, Christopher D Manning, et al. 2013. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 455–465.

Anders Søgaard and Yoav Goldberg. 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)*, volume 2, pages 231–235.

Mark Steedman. 2011. Romantics and Revolutionaries. *Linguistic Issues in Language Technology*, 6.

Pontus Stenetorp. 2013. Transition-based Dependency Parsing Using Recursive Neural Networks. In *Deep Learning Workshop at the 2013 Conference on Neural Information Processing Systems (NIPS)*.

Milan Straka, Jan Hajič, and Jana Straková. 2016. UDPipe: trainable pipeline for processing CoNLL-U files performing tokenization, morphological analysis, POS tagging and parsing. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*, pages 4290–4297.

Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. 2018. Linguistically-informed self-attention for semantic role labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5027–5038.

Sara Stymne, Miryam de Lhoneux, Aaron Smith, and Joakim Nivre. 2018. Parser Training with Heterogeneous Treebanks. In *Proceedings of the 56th Annual Meeting of the ACL, Short papers*, pages 619–625.

John Sylak-Glassman. 2016. The composition and use of the universal morphological feature schema (unimorph schema). *Johns Hopkins University*.

Oscar Täckström, Ryan McDonald, and Joakim Nivre. 2013. Target language adaptation of discriminative transfer parsers. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1061–1071.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566.

Lucien Tesnière. 1959. *Éléments de syntaxe structurale*. Editions Klincksieck.

Jörg Tiedemann. 2012. Parallel data, tools and interfaces in OPUS. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC-2012)*, pages 2214–2218.

Jörg Tiedemann. 2015. Cross-lingual dependency parsing with universal dependencies and predicted PoS labels. In *Proceedings of the Third International Conference on Dependency Linguistics (Depling 2015)*, pages 340–349.

Jörg Tiedemann, Željko Agić, and Joakim Nivre. 2014. Treebank translation for cross-lingual parser induction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 130–140.

Ivan Titov and James Henderson. 2007. A Latent Variable Model for Generative Dependency Parsing. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 144–155.

Ke Tran, Arianna Bisazza, and Christof Monz. 2018. The Importance of Being Recurrent for Modeling Hierarchical Structure. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4731–4736.

Yulia Tsvetkov, Sunayana Sitaram, Manaal Faruqui, Guillaume Lample, Patrick Littell, David Mortensen, Alan W Black, Lori Levin, and Chris Dyer. 2016. Polyglot Neural Language Models: A Case Study in Cross-Lingual Phonetic Representation Learning. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1357–1366.

Clara Vania, Andreas Grivas, and Adam Lopez. 2018. What do character-level models learn about morphology? the case of dependency parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2573–2583.

Clara Vania, Yova Kementchedjhieva, Anders Søgaard, and Adam Lopez. 2019. A systematic comparison of methods for low-resource dependency parsing on genuinely low-resource languages. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Viveka Velupillai. 2012. *An introduction to linguistic typology*. John Benjamins Publishing.

David Vilares, Carlos Gómez-Rodríguez, and Miguel A Alonso. 2016. One model, two languages: training bilingual parsers with harmonized treebanks. In

Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), volume 2, pages 425–431.

Oriol Vinyals, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey Hinton. 2015. Grammar as a foreign language. In *Advances in neural information processing systems*, pages 2773–2781.

Joe H Jr Ward. 1963. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244.

Ethan Wilcox, Roger Levy, Takashi Morita, and Richard Futrell. 2018. What do RNN Language Models Learn about Filler–Gap Dependencies? In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 211–221.

Guillaume Wisniewski and Ophélie Lacroix. 2017. A systematic comparison of syntactic representations of dependency parsing. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*, pages 146–152.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical Dependency Analysis with Support Vector Machines. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 195–206.

Zichao Yang. 2019. *Incorporating Structural Bias into Neural Networks for Natural Language Processing*. Ph.D. thesis, Carnegie Mellon University.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489.

David Yarowsky, Grace Ngai, and Richard Wicentowski. 2001. Inducing multilingual text analysis tools via robust projection across aligned corpora. In *Proceedings of the first international conference on Human language technology research*, pages 1–8.

Xiang Yu, Ngoc Thang Vu, and Jonas Kuhn. 2018. Approximate dynamic oracle for dependency parsing with reinforcement learning. In *Proceedings of the Second Workshop on Universal Dependencies (UDW 2018)*, pages 183–191.

Daniel Zeman. 2008. Reusable Tagset Conversion Using Tagset Drivers. In *Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC)*, pages 213–218.

Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov. 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.

Daniel Zeman, David Mareček, Martin Popel, Loganathan Ramasamy, Jan Štěpánek, Zdeněk Žabokrtský, and Jan Hajič. 2012. HamleDT: To parse or not to parse? In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC)*, pages 2735–2741.

Daniel Zeman, Martin Popel, Milan Straka, Jan Hajič, Joakim Nivre, Filip Ginter, Juhani Luotolahti, Sampo Pyysalo, Slav Petrov, Martin Potthast, Francis Tyers, Elena Badmaeva, Memduh Gokirmak, Anna Nedoluzhko, Silvie Cinková, Jan Hajič jr., Jaroslava Hlaváčová, Václava Kettnerová, Zdeňka Urešová, Jenna Kanerva, Stina Ojala, Anna Missilä, Christopher D. Manning, Sebastian Schuster, Siva Reddy, Dima Taji, Nizar Habash, Herman Leung, Marie-Catherine de Marneffe, Manuela Sanguinetti, Maria Simi, Hiroshi Kanayama, Valeria de Paiva, Kira Droganova, Héctor Martínez Alonso, Çağrı Çöltekin, Umut Sulubacak, Hans Uszkoreit, Vivien Macketanz, Aljoscha Burchardt, Kim Harris, Katrin Marheinecke, Georg Rehm, Tolga Kayadelen, Mohammed Attia, Ali Elkahky, Zhuoran Yu, Emily Pitler, Saran Lertpradit, Michael Mandl, Jesse Kirchner, Hector Fernandez Alcalde, Jana Strnadová, Esha Banerjee, Ruli Manurung, Antonio Stella, Atsuko Shimada, Sookyoung Kwak, Gustavo Mendonça, Tatiana Lando, Rattima Nitisaroj, and Josie Li. 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.

Daniel Zeman and Philip Resnik. 2008. Cross-language parser adaptation between related languages. In *Proceedings of the IJCNLP-08 Workshop on NLP for Less Privileged Languages*, pages 35–42.

Yuan Zhang and Regina Barzilay. 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.

Yue Zhang and Joakim Nivre. 2011. Transition-Based Parsing with Rich Non-Local Features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 188–193.

Appendix A.
ISO codes

Table 7.2. *ISO codes of UD languages*

Language	ISO	Language	ISO
Afrikaans	af	Korean	ko
Ancient Greek	grc	Kurmanji	kmr
Arabic	ar	Latin	la
Armenian	hy	Latvian	lv
Basque	eu	Lithuanian	lt
Breton	br	Marathi	mr
Bulgarian	bg	Naija	pcm
Buryat	bxr	North Sami	sme
Catalan	ca	Norwegian	no
Chinese	zh	Old Church Slavonic	cu
Coptic	cop	Old French	fro
Croatian	hr	Persian	fa
Czech	cs	Polish	pl
Danish	da	Portuguese	pt
Dutch	nl	Romanian	ro
English	en	Russian	ru
Estonian	et	Serbian	sr
Faroese	fo	Slovak	sk
Finnish	fi	Slovenian	sl
French	fr	Spanish	es
Galician	gl	Swedish	sv
German	de	Swedish Sign Language	swl
Gothic	got	Tamil	ta
Greek	el	Telugu	te
Hebrew	he	Thai	th
Hindi	hi	Turkish	tr
Hungarian	hu	Ukrainian	uk
Indonesian	id	Upper Sorbian	hsb
Irish	ga	Urdu	ur
Italian	it	Uyghur	ug
Japanese	ja	Vietnamese	vi
Kazakh	kk		

Appendix B. Non-Projective Statistics

Table 7.3 and 7.4 provide detailed statistics about non-projective arcs and trees in UD 2.1. Rows are sorted in decreasing order of percentage of non-projective arcs.

Treebank	sentences	non-proj	%	arcs	non-proj	%
Ancient Greek	11476	7201	62.8	159895	30205	18.9
Swedish Sign Language	87	24	27.6	644	70	10.9
Latin	598	245	41.0	8018	808	10.1
Basque	5396	1809	33.5	72974	5626	7.7
Ancient Greek-PROIEL	14846	5363	36.1	184382	14100	7.7
Latin-PROIEL	14192	3450	24.3	147044	8895	6.1
Latin-ITTB	15808	5939	37.6	270403	16206	6.0
Gothic	3387	648	19.1	35024	1700	4.9
English-LinES	2738	795	29.0	50095	2253	4.5
Old_Church_Slavonic	4123	742	18.0	37432	1532	4.1
Slovenian-SST	1054	115	10.9	9487	333	3.5
Hungarian	910	191	21.0	20166	706	3.5
Upper_Sorbian	23	3	13.0	460	16	3.5
Dutch	12269	1696	13.8	186046	6366	3.4
Czech-FicTree	10160	1298	12.8	133598	4483	3.4
Kazakh	31	8	25.8	529	17	3.2
Coptic	364	122	33.5	9273	292	3.2
Portuguese	8331	2227	26.7	206740	6259	3.0
Swedish-LinES	2738	554	20.2	48325	1401	2.9
Danish	4383	848	19.4	80378	2065	2.6
Czech-CLTT	860	132	15.4	26742	622	2.3
Lithuanian	153	25	16.3	3210	74	2.3
Turkish	3685	369	10.0	38082	880	2.3
Urdu	4043	882	21.8	108690	2354	2.2
Slovenian	6478	868	13.4	112530	2292	2.0
North_Sami	2257	175	7.8	16835	343	2.0
Galician-TreeGal	200	43	21.5	4855	98	2.0
Czech	68495	8027	11.7	1173282	22885	2.0
German	13814	1779	12.9	263536	5011	1.9
Afrikaans	1315	278	21.1	33894	587	1.7
Czech-CAC	23478	3108	13.2	472608	7960	1.7

Table 7.3. *Non-projective statistics in UD 2.1 – part 1*

Treebank	sentences	non-proj	%	tokens	non-proj	%
Estonian	6959	617	8.9	85567	1428	1.7
Dutch-LassySmall	6041	389	6.4	81243	1354	1.7
Hindi	13304	1839	13.8	281057	4555	1.6
Belarusian	260	31	11.9	5217	78	1.5
Finnish-FTB	14981	856	5.7	127602	1872	1.5
Norwegian-Bokmaal	15696	1218	7.8	243887	2978	1.2
Irish	121	21	17.4	3183	37	1.2
Catalan	13123	1326	10.1	417587	4595	1.1
Norwegian-Nynorsk	14174	1128	8.0	245330	2669	1.1
Romanian	8043	911	11.3	185113	1923	1.0
Greek	1662	213	12.8	42326	442	1.0
Slovak	8483	282	3.3	80575	841	1.0
Finnish	12217	750	6.1	162621	1665	1.0
Ukrainian	4506	306	6.8	75054	769	1.0
Marathi	373	20	5.4	2997	28	0.9
Korean	4400	302	6.9	52328	474	0.9
Latvian	4124	215	5.2	62254	529	0.9
Russian	3850	297	7.7	75964	630	0.8
Spanish-AnCora	14305	1361	9.5	444617	3415	0.8
Croatian	7689	656	8.5	169283	1258	0.7
French-FTB	14759	1543	10.5	455946	3278	0.7
Russian-SynTagRus	48814	3090	6.3	870034	5652	0.7
French	14554	1259	8.7	356613	2190	0.6
Arabic-NYUAD	15789	1216	7.7	590819	3622	0.6
Arabic	6075	654	10.8	223881	1357	0.6
Persian	4798	329	6.9	121064	715	0.6
English	12543	583	4.7	204585	1164	0.6
Romanian-Nonstandard	621	23	3.7	10352	53	0.5
Spanish	14187	1012	7.1	382436	1901	0.5
Vietnamese	1400	43	3.1	20285	88	0.4
Bulgarian	8907	277	3.1	124336	535	0.4
Italian	12838	555	4.3	270703	1109	0.4
Indonesian	4477	220	4.9	97531	403	0.4
French-ParTUT	803	49	6.1	24123	91	0.4
Serbian	2935	117	4.0	65764	239	0.4
Portuguese-BR	9664	549	5.7	255755	909	0.4
English-ParTUT	1781	79	4.4	43518	151	0.4
Swedish	4303	137	3.2	66645	210	0.3
Tamil	400	12	3.0	6329	18	0.3
French-Sequoia	2231	60	2.7	50561	120	0.2
Italian-ParTUT	1781	53	3.0	48934	83	0.2
Japanese	7164	195	2.7	161900	262	0.2
Polish	6100	52	0.9	62501	90	0.1
Hebrew	5241	66	1.3	137680	126	0.1
Italian-PoSTWITA	2808	26	0.9	51606	40	0.1
Chinese	3997	25	0.6	98608	57	0.1
Telugu	1051	1	0.1	5082	1	0.0
Galician	2272	0	0.0	79327	0	0.0

Table 7.4. *Non-projective statistics in UD 2.1 – part 2*

Appendix C. AVC Representation Results with MaltParser

Table 7.5 gives the results of our experiments in de Lhoneux and Nivre (2016).

Table 7.5. *LAS with the 4 different pipelines and difference in accuracy between pairs. Bold indicates the best between the first and second column. Italics indicates the best between the first and third column. Significance for the McNemar test between* UD2UD *and* MS2UD *as well as between* MS2MS *and* UD2MS *is indicated with* $* = p < .05$ *and* $** = p < .01$.

Language	UD2UD	MS2UD	δ	MS2MS	UD2MS	δ
Basque	***64.4***	63.8**	-0.6	64.0	64.4	0.4
Bulgarian	***83.4***	83.2*	-0.2	82.5	82.9	0.4
Croatian	***75.9***	74.6**	-1.3	73.7	75.9	2.2
Czech	***80.0***	76.5**	-3.5	76.4	79.9	3.5
Danish	***75.9***	75.2**	-0.7	74.8	75.8	1.0
English	***81.7***	80.4**	-1.3	80.2	81.5	1.3
Estonian	77.1	**77.8**	0.7	*77.6*	77.0	-0.6
Finnish	***66.9***	66.4*	-0.5	65.9	66.4	0.5
Finnish-FTB	**71.3**	70.4**	-0.9	*72.1*	72.5	0.4
French	***82.1***	81.6**	-0.5	81.3	81.8	0.5
German	***76.6***	76.0**	-0.6	75.4	76.1	0.7
Greek	*75.2*	**75.3**	0.1	75.1	75.2	0.1
Hebrew	***78.4***	77.9**	-0.5	77.9	78.5	0.6
Hindi	***85.4***	84.2**	-1.2	84.9	85.2	0.3
Italian	***83.8***	83.6	-0.2	83.3	83.6	0.3
Norwegian	***84.5***	82.0**	-2.5	81.7	84.5	2.8
Old_Church_Slavonic	***68.8***	68.7	-0.1	68.7	68.9	0.2
Persian	***81.1***	79.8**	-1.3	79.8	81.1	1.3
Polish	***79.4***	79.1	-0.3	79.0	79.3	0.3
Portuguese	81.3	**81.5**	0.2	*81.6*	81.3	-0.3
Romanian	***64.2***	62.5*	-1.7	64.0	64.6	0.6
Slovenian	***80.8***	79.7**	-1.1	79.8	80.8	1.0
Spanish	***81.5***	81.2**	-0.3	81.2	81.4	0.2
Swedish	***76.8***	75.7**	-1.1	75.6	76.7	1.1
Tamil	**67.2**	67.1	-0.1	*67.4*	67.5	0.1
Average	***76.9***	76.2	-0.8	76.2	76.9	0.8

Appendix D.
Task results for NFMV-MS and MAUX-UD

Table 7.6. *Classification accuracy of the majority baseline (maj) and classifier trained on the token (tok), type and character (char) vectors of FMVs, NFMVs and MAUX on agreement (A) and transitivity (T) tasks. Difference (δ) to majority baseline of these classifiers. Average differences significantly higher than the baseline with $p < .05$ are in bold and with $p < .01$ are in bold and italics.*

		AVC	NFMV-MS			MAUX-UD			δ nfmv			δ maux		
		maj	tok	type	char	tok	type	char	tok	type	char	tok	type	char
T	ca	66.9	88.9	78.8	73.7	87.9	67.2	67.4	22.1	11.9	6.8	21.0	0.3	0.5
	fi	49.1	82.2	72.3	74.0	77.4	61.9	59.1	33.1	23.1	24.8	28.2	12.8	10.0
	hr	51.2	81.2	75.4	70.4	76.4	55.0	55.6	30.0	24.2	19.3	25.3	3.9	4.4
	nl	70.5	89.2	77.2	72.0	89.7	80.6	81.9	18.7	6.7	1.5	19.2	10.1	11.4
	av.	59.4	85.4	75.9	72.5	82.8	66.2	66.0	***26.0***	**16.5**	**13.1**	***23.4***	**6.8**	**6.6**
	sd	10.8	4.3	2.8	1.6	6.9	10.8	11.7	6.7	8.6	10.8	4.1	5.7	5.0
A	ca	76.0	76.0	76.0	77.8	86.8	96.3	97.3	0.0	0.0	1.8	10.8	20.3	21.3
	fi	67.5	69.4	67.5	71.1	77.6	89.4	88.8	1.9	0.0	3.6	10.1	21.9	21.3
	hr	71.2	73.5	72.7	71.2	71.2	95.3	94.9	2.3	1.5	0.0	0.0	24.0	23.7
	nl	72.4	75.1	73.5	72.4	86.2	93.9	94.7	2.8	1.1	0.0	13.8	21.6	22.3
	av.	71.8	73.5	72.4	73.1	80.4	93.7	93.9	**1.7**	0.7	1.3	**8.7**	***21.9***	***22.2***
	sd	3.5	2.9	3.6	3.2	7.4	3.1	3.6	1.2	0.8	1.7	6.0	1.6	1.1

Appendix E. CoNLL 2018 Shared Task Details

We use two different types of parsing models:

1. Single treebank models, where we train one model per treebank (17 models applied to 18 treebanks, including special models for Breton KEB and Thai PUD).
2. Multi-treebank models
   - Monolingual models, based on multiple treebanks for one language (4 models, trained on 10 treebanks, applied to 11 treebanks).
   - Multilingual models, based on treebanks from several (mostly) closely related languages (12 models, trained on 48 treebanks, applied to 52 treebanks; plus a special model for Naija NSC).

The two possible representation of words at the input of the word BiLSTM are therefore:

$$x_i = [pe(w_i); ce(w_i); e(t_i)] \tag{7.1}$$

$$x_i = [pe(w_i); ce(w_i); e(t_i); tb_e(w_i)] \tag{7.2}$$

When a multi-treebank model is applied to a test set from a treebank with training data, we naturally use the treebank embedding of that treebank also for the test sentences. However, when parsing a test set with no corresponding training data, we have to use one of the other treebank embeddings. I henceforth refer to the treebank selected for this purpose as the *proxy* treebank (or simply *proxy*). In order to keep the training times and language balance in each model reasonable, we cap the number of sentences used from each treebank to 15,000, with a new random sample selected at each epoch. This only affects a small number of treebanks, since most training sets are smaller than 15,000 sentences. For all our multi-treebank models, we apply the treebank embeddings described earlier. Where two or more treebanks in a multilingual model come from the same language, we use separate treebank embeddings for each of them.

For the nine treebanks that have no training data we use different strategies:

- For Japanese Modern, we apply the mono-treebank Japanese GSD model.

- For the four PUD treebanks, we apply the multi-treebank models trained using the other treebanks from that language, with the largest available treebank as proxy (except for Finnish, where we prefer Finnish TDT over FTB because TDT contains a mix of domains and FTB only contains grammar examples).
- For Faroese, we apply the model for the Scandinavian languages, which are closely related, with Norwegian Nynorsk as proxy as we believe this to be the most closely related. In addition, we map the Faroese characters {Íýúð}, which do not occur in the other Scandinavian languages, to {Iyud}.
- For Naija, an English-based creole, whose treebank according to the README file contains spoken language data, we train a special multilingual model on English EWT and the three small spoken treebanks for French, Norwegian, and Slovenian, and used English EWT as proxy.[1]
- For Thai and Breton, we create multilingual models trained with word and POS embeddings only (i.e., no character models or treebank embeddings) on Chinese and Irish, respectively. These models make use of multilingual word embeddings provided with Facebook's MUSE multilingual embeddings (Lample et al., 2018),[2] as described in more detail below.

In the case where we have a very low-resource language (31 sentences or less), we construct a multilingual model but with less related languages than for our other multilingual models. We believe this will still be stronger than a monolingual baseline which is very low with such little amount of data. For Buryat, Uyghur and Kazakh, we trained a model with Turkish, and for Kurmanji, we trained a model with Persian. For Armenian, which has only 50 training sentences, we could not find a close enough language, and instead train a single model on the available data. For all multi-treebank models, we choose the model from the epoch that has the best mean LAS score among the treebanks that have available development data. This means that treebanks without development data rely on a model that is good for other languages in the group. In the cases of the mono-treebank Armenian and Irish models, where there is no development data, we choose the model from the final training epoch. This also applies to the Breton model trained on Irish data.

Thai–Chinese

For the Thai model trained on Chinese, we were able to map Facebook's monolingual embeddings for each language to English using MUSE, thus creating multilingual Thai-Chinese embeddings. We then trained a monolingual parser model using the mapped Chinese embeddings to initialise all word embeddings, and ensuring that these were not updated during training (unlike in the standard parser setup used in the rest of our experiments in this thesis). At

[1] We had found this combination to be useful in preliminary experiments where we tried to parse French Spoken without any French training data.

[2] `https://github.com/facebookresearch/MUSE`

test time, we look up all OOV word types, which are the great majority, in the mapped Thai embeddings first, otherwise assign them to a learned OOV vector. Note that in this case, we had to increase the word embedding dimension in our parser to 300 to accomodate the larger Facebook embeddings.

Breton–Irish
For Breton and Irish, the Facebook software does not come with the necessary resources to map these languages into English. Here we instead created a small dictionary by using all available parallel data from OPUS (Tiedemann, 2012) (Ubuntu, KDE and Gnome, a total of 350K text snippets), and training a statistical machine translation model using Moses Koehn et al. (2007). From the lexical word-to-word correspondences created, we kept all cases where the translation probabilities in both directions were at least 0.4 and the words were not identical (in order to exclude a lot of English noise in the data), resulting in a word list of 6027 words. We then trained monolingual embeddings for Breton using word2vec on Wikipedia data, and mapped them directly to Irish using MUSE. A parser model was then trained, similarly to the Thai-Chinese case, using Irish embeddings as initialization, turning off updates to the word embeddings, and applying the mapped Breton embeddings at test time.

Note that after the test phase was over, we discovered that we had used a non-permitted resource when developing the UPOS tagger for Thai PUD (see Section 4 in Smith et al. (2018a)). This does not affect the ranking for any of the three scores, as confirmed by the shared task organisers. We report the official score and corrected scores in Smith et al. (2018a).