Viability of sequence labeling encodings for dependency parsing

MICHALINA STRZYŻ

Doctoral Thesis 2021

Advisors:

CARLOS GÓMEZ-RODRÍGUEZ
DAVID VILARES CALVO

Ph.D. degree in Computer Science

UNIVERSIDADE DA CORUÑA

CARLOS GÓMEZ-RODRÍGUEZ, Associate Professor at the Department of Computer Science and Information Technologies at the University of A Coruña,

DAVID VILARES CALVO, Assistant Professor at the Department of Computer Science and Information Technologies at the University of A Coruña,

hereby certify

that the dissertation entitled *Viability of sequence labeling encodings for dependency parsing*, submitted to Universidade da Coruña by Michalina Strzyż, has been carried out under our supervision and meets all the requirements for the award of *Ph.D. degree*.

CARLOS GÓMEZ-RODRÍGUEZ
First advisor

DAVID VILARES CALVO
Second advisor

Acknowledgements

The path to become a researcher felt to me like a sine wave with exciting points but also arduous ones, reflecting the true nature of what it takes to push the boundaries of Science. Research, however, is not a one man show and this thesis would not be possible without people with whom I worked alongside.

Firstly, I would like to express my sincere gratitude to Carlos Gómez-Rodríguez for giving me the opportunity to pursue the PhD under his advisory. Carlos has provided me with an excellent ground for unlimited learning, as well as academic and personal growth. I want to thank Carlos for believing in me, something that is incredibly important and encouraging for an aspiring young researcher. I will always look up to Carlos for being both an outstanding NLP researcher with ingenious ideas, as well as a great advisor who helps his students to excel and nurtures their enthusiasm for science.

Secondly, I am grateful to David Vilares with whom I was fortunate to work closely from the beginning of my PhD. David has been very supportive and was always nearby with helpful advice. I want to thank David for being a great advisor and for teaching me how to do research of the highest standards.

Furthermore, I would like to thank the LyS group for the relaxing lunches and insightful discussions.

On a more personal front, I wish to thank my closest family and friends. I am particularly grateful to Rui for always being there for me and for his unconditional support and love. I want to thank Domi, for all the encouragement, motivation and long talks, as well as Nina, the smallest one, for sharing her pure joy of life with me every day. I was also fortunate to meet the Chaflán group that has shown me the Galician way of living and the Trupki group that always takes part in my everyday life despite the distance.

Finally, this work has been carried out thanks to the funding from the European Research Council (ERC), under the European Union's Horizon 2020 research and innovation programme (FASTPARSE, grant agreement No 714150).

Abstract

This thesis presents new methods for recasting dependency parsing as a sequence labeling task yielding a viable alternative to the traditional transition- and graph-based approaches. It is shown that sequence labeling parsers provide several advantages for dependency parsing, such as: (i) a good trade-off between accuracy and parsing speed, (ii) genericity which enables running a parser in generic sequence labeling software and (iii) pluggability which allows using full parse trees as features to downstream tasks.

The backbone of dependency parsing as sequence labeling are the encodings which serve as linearization methods for mapping dependency trees into discrete labels, such that each token in a sentence is associated with a label. We introduce three encoding families comprising: (i) head selection, (ii) bracketing-based and (iii) transition-based encodings which are differentiated by the way they represent a dependency tree as a sequence of labels. We empirically examine the viability of the encodings and provide an analysis of their facets.

Furthermore, we explore the feasibility of leveraging external complementary data in order to enhance parsing performance. Our sequence labeling parser is endowed with two kinds of representations. First, we exploit the complementary nature of dependency and constituency parsing paradigms and enrich the parser with representations from both syntactic abstractions. Secondly, we use human language processing data to guide our parser with representations from eye movements.

Overall, the results show that recasting dependency parsing as sequence labeling is a viable approach that is fast and accurate and provides a practical alternative for integrating syntax in NLP tasks.

Resumen

Esta tesis presenta nuevos métodos para reformular el análisis sintáctico de dependencias como una tarea de etiquetado secuencial, lo que supone una alternativa viable a los enfoques tradicionales basados en transiciones y grafos. Se demuestra que los analizadores de etiquetado secuencial ofrecen varias ventajas para el análisis sintáctico de dependencias, como por ejemplo (i) un buen equilibrio entre la precisión y la velocidad de análisis, (ii) la genericidad que permite ejecutar un analizador en un software genérico de etiquetado secuencial y (iii) la conectividad que permite utilizar el árbol de análisis completo como características para las tareas posteriores.

El pilar del análisis sintáctico de dependencias como etiquetado secuencial son las codificaciones que sirven como métodos de linealización para transformar los árboles de dependencias en etiquetas discretas, de forma que cada *token* de una frase se asocia con una etiqueta. Introducimos tres familias de codificación que comprenden: (i) selección de núcleos, (ii) codificaciones basadas en corchetes y (iii) codificaciones basadas en transiciones que se diferencian por la forma en que representan un árbol de dependencias como una secuencia de etiquetas. Examinamos empíricamente la viabilidad de las codificaciones y ofrecemos un análisis de sus facetas.

Además, exploramos la viabilidad de aprovechar datos complementarios externos para mejorar el rendimiento del análisis sintáctico. Dotamos a nuestro analizador sintáctico de dos tipos de representaciones. En primer lugar, explotamos la naturaleza complementaria de los paradigmas de análisis sintáctico de dependencias y constituyentes, enriqueciendo el analizador sintáctico con representaciones de ambas abstracciones sintácticas. En segundo lugar, utilizamos datos de procesamiento del lenguaje humano para guiar nuestro analizador con representaciones de los movimientos oculares.

En general, los resultados muestran que la reformulación del análisis sintáctico de dependencias como etiquetado de secuencias es un enfoque viable, rápido y preciso, y ofrece una alternativa práctica para integrar la sintaxis en las tareas de PLN.

Resumo

Esta tese presenta novos métodos para reformular a análise sintáctica de dependencias como unha tarefa de etiquetaxe secuencial, o que supón unha alternativa viable aos enfoques tradicionais baseados en transicións e grafos. Demóstrase que os analizadores de etiquetaxe secuencial ofrecen varias vantaxes para a análise sintáctica de dependencias, por exemplo (i) un bo equilibrio entre a precisión e a velocidade de análise, (ii) a xenericidade que permite executar un analizador nun software xenérico de etiquetaxe secuencial e (iii) a conectividade que permite empregar a árbore de análise completa como características para as tarefas posteriores.

O piar da análise sintáctica de dependencias como etiquetaxe secuencial son as codificacións que serven como métodos de linealización para transformar as árbores de dependencias en etiquetas discretas, de forma que cada *token* dunha frase se asocia cunha etiqueta. Introducimos tres familias de codificación que comprenden: (i) selección de núcleos, (ii) codificacións baseadas en corchetes e (iii) codificacións baseadas en transicións que se diferencian pola forma en que representan unha árbore de dependencia como unha secuencia de etiquetas. Examinamos empíricamente a viabilidade das codificacións e ofrecemos unha análise das súas facetas.

Ademais, exploramos a viabilidade de aproveitar datos complementarios externos para mellorar o rendemento da análise sintáctica. O noso analizador sintáctico de etiquetaxe secuencial está dotado de dous tipos de representacións. En primeiro lugar, explotamos a natureza complementaria dos paradigmas de análise sintáctica de dependencias e constituíntes e enriquecemos o analizador sintáctico con representacións de ambas abstraccións sintácticas. En segundo lugar, empregamos datos de procesamento da linguaxe humana para guiar o noso analizador con representacións dos movementos oculares.

En xeral, os resultados mostran que a reformulación da análise sintáctico de dependencias como etiquetaxe de secuencias é un enfoque viable, rápido e preciso, e ofrece unha alternativa práctica para integrar a sintaxe nas tarefas de PLN.

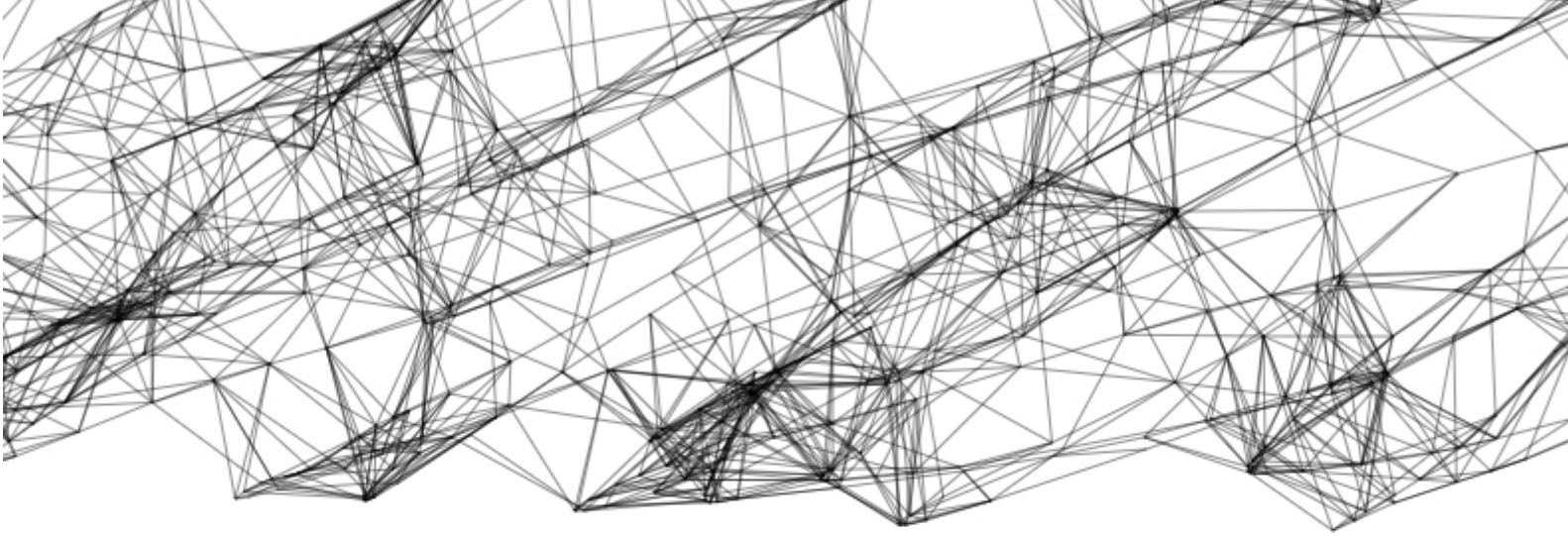
Acronyms

BERT	Bidirectional Encoder Representations from Transformers
BiLSTM	Bidirectional Long Short Term Memory Networks
LAS	Labeled Attachment Score
MLP	Multi-Layer Perceptron
MTL	Multi-Task Learning
NLP	Natural Language Processing
UAS	Unlabeled Attachment Score
UD	Universal Dependencies
PoS	Part-of-Speech
P	Precision
R	Recall
SOTA	state-of-the-art

Contents

List of Figures

List of Tables



Part I

INTRODUCTION AND PRELIMINARIES

1 Introduction

1.1 Motivation

Syntactic parsing has been drawing attention for the past decades and became an inherent foundation block in the Natural Language Processing (NLP) field. Its goal is to automatically determine the syntactic structure of a sentence. One of the most studied parsing paradigms is dependency parsing which provides knowledge about the interplay of dependencies and relationship between words that serve as a useful abstraction for representing and processing human language. Employing syntactic features has been shown to be beneficial for a range of NLP tasks. Dependency parsing has most recently found application, for instance, in: machine translation (Zhang et al., 2019), question answering (Cao et al., 2021), semantic role labeling (Strubell et al., 2018), summarization (Song, Zhao, and Liu, 2018), opinion role labeling (Zhang et al., 2020) or dialogue generation (Zhou et al., 2018).

Recent neural network advances have enabled an efficient syntactic feature extraction and have provided dependency parsers that yield very accurate results on certain domains. However, since dependency parsing is one of the core NLP tasks on which other downstream tasks rely, there are manifold aspects beyond the accuracy to consider when combining a parser with other NLP applications. One of the crucial assets of a dependency parser that emerges in the era of massive data is the parser's ability to maintain a good *trade-off* between the parsing speed and accuracy in order to mitigate the recurrent bottleneck of parsing speed that has a negative effect on the pipeline. To do so, one may leverage the capabilities of neural networks to propose alternative approaches in dependency parsing based on *simpler* methods than the existing ones to improve parser's computational efficiency while achieving a competitive accuracy. Additionally, it may be beneficial that a parser relies on *generic* methods, so that it can be used as an off-the-shelf software and benefit from its upstream improvements. Moreover, a parser may obtain additional gains from leveraging external *complementary data* (for instance, data used for training other downstream sequence labeling tasks) without sacrificing its efficiency and no need for any additional ad-hoc methods to do so. Hence, this thesis will pay closer attention to these aspects and state some key questions that we believe are substantial in the context of further enhancements.

DEPENDENCY PARSING DURING THE NEURAL NETWORKS HEYDAY

Traditionally, there have been two dominant approaches in dependency parsing: *transition-* (Nivre, 2003) and *graph-based* (McDonald, Crammer, and Pereira, 2005). These are among the most widely used while their facets are well-studied. In recent years, artificial neural networks seem to have revolutionized the NLP field and many tasks have excelled due to the powerful neural architectures yielding state-of-the-art (SOTA) results. Likewise, there have been early attempts to enhance dependency parsers with the neural network methods (Titov and Henderson, 2007), however neural dependency parsing has become remarkably prevalent with the work of Chen and Manning (2014) that successfully employed a feed-forward network in a transition-based parser. Besides achieving great performance, it has been shown that the neural approach mitigates the manual feature extraction efforts that formerly used to be crucial and challenging. Thereby, in the light of the capabilities of neural networks, many syntactic parsers became conceptually simpler than their antecedents. Models with a large feature set (Zhang and Nivre, 2011) were simplified by relying on a few core features (Chen and Manning, 2014). Thereafter, as succeeding neural architectures have been proposed, Bidirectional Long Short Term Memory Networks (BiLSTM) (Graves and Schmidhuber, 2005) have been shown to notably suit the dependency parsing task and enabled reducing the feature engineering efforts even further (Kiperwasser and Goldberg, 2016). Eventually, a minimal set of two features has been shown to suffice to obtain competitive results (Shi, Huang, and Lee, 2017). For the past years, the BiLSTM-based parsers obtained SOTA performance (Kiperwasser and Goldberg, 2016; Dozat and Manning, 2017), although recently (at the time of writing this thesis in 2021), pointer networks (Vinyals, Fortunato, and Jaitly, 2015) have been successfully applied to dependency parsing (Ma et al., 2018; Fernández-González and Gómez-Rodríguez, 2019) and there has also been a noticeable rise in the use of transformers (Vaswani et al., 2017) as a promising alternative (Zhou and Zhao, 2019; Mrini et al., 2019).

Another line of research attempts to recast dependency parsing as a divergent problem obviating the use of the traditional transition- or graph-based algorithms, structural features or even the requirement of the output having a tree structure. For instance, Zhang, Cheng, and Lapata (2017) define dependency parsing as a head selection task where the problem is reduced to independently finding the most probable head for each token in a sentence using BiLSTMs. In the same vein, Li et al. (2018) propose a sequence to sequence dependency parser that predicts the relative position of a head for a given word. Furthermore, Spoustová and Spousta (2010) attempt to recast dependency parsing as a sequence labeling task, where a dependency tree is represented as a sequence of

labels and each word is associated with a single label. However, their approach relies on a non-neural model and achieves inferior accuracy than the counterpart parsers. Recently, Gómez-Rodríguez and Vilares (2018) show however that constituency parsing as sequence labeling can be successfully employed with neural network architectures and provides a competitive performance.

Hence, the aim of this thesis is to call into question whether in the age of neural advances, reducing dependency parsing to a sequence labeling task enhanced with novel linearization methods may serve as a viable alternative to notably more complex systems. One of the main advantages of this approach is that it enables opting for architectural simplicity that helps increase the computational efficiency of dependency parsing.

RECURRING BOTTLENECK OF PARSING SPEED

The past decades of parsing research gave rise to a myriad of approaches that have been pushing the boundaries of dependency parsers' performance. However, it has been often opted for the accuracy gains rather than the computational efficiency of a parser. This may be particularly precarious in the context of massive data that is available today, where the need of efficient parsers emerges since systems that are computationally expensive may in fact impede feasible and scalable parsing (Gómez-Rodríguez, 2017). A discrepancy between the accuracy and the efficiency of a parser may have further negative implications. For instance, very accurate but inefficient parsers may be intractable for embedding them in other NLP applications. More particularly, Gómez-Rodríguez, Alonso-Alonso, and Vilares (2019) show that the difference in parser's accuracy used in the sentiment analysis task does not affect proportionally the performance of the end task while the computational burden varies among them.

Furthermore, the bottleneck of parsing efficiency is even more prominent in the age of deep neural networks, where dependency parsers increasingly rely on models with large parameter space and that require costly training entailing a considerable carbon footprint. Lately, considerable attention has been directed towards the environmental concerns around the computational cost of NLP models (Strubell, Ganesh, and McCallum, 2019; Schwartz et al., 2019) motivating the attempts of providing smaller and more efficient models that yield similar accuracy (Jiao et al., 2020; Wang et al., 2020).

Thus, a question arises whether more attention should be aimed at proposing dependency parsers that maintain the speed-accuracy equilibrium. Some parts of this thesis seek to determine whether the dissonance between the parsing accuracy and the speed of dependency parsers can be overcome by a sequence labeling parser that is based on a simple and generic approach. Furthermore, this method may also fa-

cilitate integrating representations from external complementary data.

REPRESENTATIONAL ENRICHMENT FROM COMPLEMENTARY DATA

The learning process of a parser does not need to solely rely on data with dependency annotation. There are numerous types of external complementary data which provide some notion of word relations in a sentence and yield syntactic representations beyond those captured by the dependency grammar. Hence, the aim of our work is to enrich the representations acquired by the sequence labeling parser with complementary data by means of auxiliary tasks in a Multi-Task Learning (MTL) setup.

An example of complementary data that a dependency parser may leverage is, for instance, a constituency treebank. Both dependency (Mel'cuk, 1988) and constituency grammar (Chomsky, 1956) are parallel formalisms expressing some notion of syntactic structures. The former represents a sentence as a tree with head-dependent relation for each word pair, while in the latter, a constituency tree reflects word relation with respect to the position in the phrasal hierarchy. Hence, they are likely to contain some complementary syntactic representations that may enrich both types of parsers. Traditionally, the improvements of dependency and constituency parsers used to be mostly carried out independently. There were few attempts of incorporating representations from both formalisms, however they resulted in arguably complex parsers (Klein and Manning, 2002; Ren, Chen, and Kit, 2013). Hence, in this thesis we aim to investigate whether it is feasible to learn dependency and constituency abstractions in order to obtain a more accurate parser without sacrificing its parsing speed.

Another promising research line is directed towards the use of human language processing data in the NLP field. Recent work shows that measures reflecting some human cognitive processes may be beneficial for NLP tasks (Barrett, 2018; Hollenstein et al., 2019). For instance, a cognitive skill such as reading seems particularly relevant for dependency parsing, since it provides some insight into how much time a reader spends on some core parts of the sentence, such as subject etc. Hence, this thesis also explores the impact of learning eye movement measures and whether eye-tracking data may improve the performance of our dependency parser.

1.2 Task definition and contributions

We turn now to the task definition and following the canonical work of Spoustová and Spousta (2010), recasting dependency parsing as sequence labeling consists in associating each token with a discrete label obtained with a linearization method for dependency tree encoding. To

make it practical under the neural network setup, we formalise the task following Gómez-Rodríguez and Vilares (2018).

More formally, given a set of trees T for sentences of length $|w|$ and a set of categorical labels L, we define an encoding function $\psi : T_{|w|} \rightarrow L^{|w|}$ that transforms a tree $t \in T$ to a sequence of discrete labels. Conversely, in order to parse a sentence, a sequence labeling parser processes a sentence with $|w|$ tagging actions by predicting for each word w_i a corresponding label l_i. The predicted sequence of labels can be then decoded to a dependency tree. Due to non-surjectivity, some of the labels in the sequence may be erroneously predicted resulting in tokens with dislocated or missing heads. In such a case, a postprocessing step is required in order to ensure a well-formed output tree.

This thesis introduces several enhancements to dependency parsing, whose scope and contributions can be outlined as follows. We propose:

- A sequence labeling parser whose main asset is accurate and fast dependency parsing.
- Several linearization methods that serve as the mainstay of the dependency parsing as sequence labeling approach enabling encoding dependency trees as a sequence of labels. The encodings are grouped into three families that differ in how they transform a tree structure into a sequence of labels. The first encoding family is based on *head selection*, where the head is encoded in a label through its absolute or the relative position. The second family consists of *bracketing-based* encodings, in which dependency arcs are represented as pairs of matching brackets. This includes methods for increasing coverage of non-projectivity by applying the property of 2-planarity. The third family is *transition-based*, where a label contains a subsequence of transitions retrieved from the transition-based algorithms, resulting from a unifying theory of transition-based and sequence labeling parsing.
- In-depth analyses of each encoding family in terms of their strengths and limitations based on the empirical evaluations.
- Methods for leveraging complementary data, such as constituency analysis and eye movement measures in dependency parsing as sequence labeling by means of auxiliary tasks in a MTL setup.

1.3 Thesis outline

This thesis is divided into four central parts that are based on a collection of published work (Section 1.4).

Part I outlines the topic and the scope of this thesis followed by an overview of the essential preliminaries to dependency parsing.

CHAPTER 1 describes the motivation behind this work and states the key questions regarding some underexplored aspects in dependency parsing.

CHAPTER 2 provides preliminary definitions and a description of resources used in dependency parsing, followed by a discussion on the shift towards neural approaches in dependency parsing. The chapter concludes with an overview of parsing speeds and neural methods used in this work.

Part II provides a formalisation of the three encoding families and includes in-depth analyses based on the evaluations. It also discusses each encoding with respect to its strengths and limitations.

CHAPTER 3 introduces a set of encodings belonging to the head selection family and examines the encodings' performance and facets.

CHAPTER 4 describes the bracketing-based encodings including a variant that relies on the property of 2-planarity in order to increase the coverage of non-projectivity. The chapter concludes with an analysis of strengths and shortcomings of the 1- and 2-planar encodings.

CHAPTER 5 presents the transition-based family of encodings obtained using novel methods for mapping transition-based parsers to sequence labeling parsers.

Part III addresses the use of external complementary data in a sequence labeling parser. In particular, it examines the leverage of constituency analyses and eye movement measures learned as auxiliary tasks in a MTL setup.

CHAPTER 6 describes a case study where dependency and constituency representations are learned jointly by a sequence labeling parser. The two parsing abstractions are used in a two-fold manner. First, the counterpart paradigm learned as an auxiliary task serves to improve the performance of the other paradigm and vice versa. Secondly, the two paradigms are learned as main tasks providing a single model that is capable of parsing both of them.

CHAPTER 7 presents a case study where a sequence labeling parser is enhanced with human language processing data. It examines whether eye movement measures learned as auxiliary tasks may be beneficial in dependency parsing.

Final conclusions are drawn in **Part IV**, where we also allude to potential directions for the future work on dependency parsing as sequence labeling.

1.4 Publications

This thesis is built upon the following collaborative conference publications:

A. Michalina Strzyz, David Vilares, and Carlos Gómez-Rodríguez (June 2019c). «Viable Dependency Parsing as Sequence Labeling.» In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 717–723. DOI: 10.18653/v1/N19-1077. URL: https://www.aclweb.org/anthology/N19-1077

B. Michalina Strzyz, David Vilares, and Carlos Gómez-Rodríguez (July 2019a). «Sequence Labeling Parsing by Learning across Representations.» In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, pp. 5350–5357. DOI: 10.18653/v1/P19-1531. URL: https://www.aclweb.org/anthology/P19-1531

C. Michalina Strzyz, David Vilares, and Carlos Gómez-Rodríguez (Nov. 2019b). «Towards Making a Dependency Parser See.» In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, pp. 1500–1506. DOI: 10.18653/v1/D19-1160. URL: https://www.aclweb.org/anthology/D19-1160

D. Michalina Strzyz, David Vilares, and Carlos Gómez-Rodríguez (Dec. 2020). «Bracketing Encodings for 2-Planar Dependency Parsing.» In: *Proceedings of the 28th International Conference on Computational Linguistics*. Barcelona, Spain (Online): International Committee on Computational Linguistics, pp. 2472–2484. URL: https://www.aclweb.org/anthology/2020.coling-main.223

E. Carlos Gómez-Rodríguez, Michalina Strzyz, and David Vilares (Dec. 2020). «A Unifying Theory of Transition-based and Sequence Labeling Parsing.» In: *Proceedings of the 28th International Conference on Computational Linguistics*. Barcelona, Spain (Online): International Committee on Computational Linguistics, pp. 3776–3793. URL: https://www.aclweb.org/anthology/2020.coling-main.336

F. David Vilares, Michalina Strzyz, Anders Søgaard, and Carlos Gómez-Rodríguez (2020). «Parsing as Pretraining.» In: *Proceed-*

ings of the Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, New York, NY, USA, February 7-12, 2020. AAAI Press, pp. 9114–9121. DOI: https://doi.org/10.1609/aaai.v34i05.6446. URL: https://aaai.org/ojs/index.php/AAAI/article/view/6446

1.5 Software

The source code for reproducibility of the results can be found at the following URLs:

- Sequence labeling parser with An Open-source Neural Sequence Labeling Toolkit (NCRF++)
  https://github.com/mstrise/dep2label
- Sequence labeling parser with Bidirectional Encoder Representations from Transformers (BERT)
  https://github.com/mstrise/dep2label-bert

2 Preliminaries

This chapter gives an overview over the essential concepts that our work is built on. First, we will recall some definitions of dependency structure and dependency parsing. Next, we will provide a short description of the selected dependency treebanks that are used to carry out our experiments and an outline of the evaluation methods. Afterwards, we will discuss the noticeable shift towards neural approach in dependency parsing and we will provide a comparison of parsing speeds of recent neural dependency parsers. Lastly, we will introduce the sequence labeling task that consists in assigning a label to each input token, and multi-task learning that enables a model to learn several tasks jointly and leverage their shared representations.

2.1 Dependency formalism

Dependency parsing relies on dependency grammar whose theory is often referred to the work of Tesnière (1959) and Mel'cuk (1988). Its principles say that the words in a sentence are linked with binary asymmetric relations and each of such word pairs consists of a *head* word governing a *dependent* word, associated with a certain *dependency relation type*. This group of dependencies within a sentence forms a *dependency tree* as illustrated in Figure 1. In the parsing literature, an artificial root node is commonly added as a root of such a tree. Hence, dependency parsing is a task of assigning automatically such a dependency structure to any given sentence.

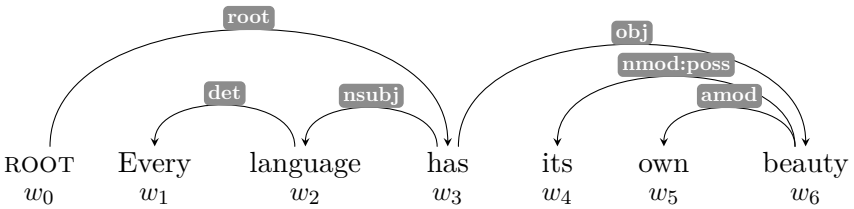


Figure 1: An example of a dependency tree.

2.1.1 Dependency tree

More formally, a dependency tree for a sentence $w = w_1, w_2, ..., w_n$ is a directed graph $G = (V, A)$, where $V = \{0...n\}$ stands for tree nodes

while A denotes dependency arcs. The vertex at index 0 represents the artificial root node that is added to act as a head for the sentence. Each arc is a triple $(h, l, d) \in A$, where $h \in V$ is the index of the head node, l the dependency type and $d \in V \setminus \{0\}$ the index of the dependent node, such that $w_h \xrightarrow{l} w_d$.

A dependency graph forms a valid dependency tree only if it is well-formed by fulfilling some conditions. Namely, each node has only one incoming arc, i.e. each word is governed by a single head, and a dependency graph can not contain cycles. These properties can be formalised as:

- SINGLE HEAD if $w_h \rightarrow w_d$ then $\nexists w_{h'} : w_{h'} \rightarrow w_d$ for $w_{h'} \neq w_h$
- ACYCLICITY if $\nexists$path: $w_h \rightarrow ... \rightarrow w_d$ and $w_d \rightarrow ... \rightarrow w_h$

On these grounds, the dependency parsing can be defined as a problem of mapping an input sentence to a dependency graph while preserving the properties described above (Kübler, McDonald, and Nivre, 2009).

2.1.2 Non-projectivity

Some languages are prone to contain syntactic structures in which dependency arcs happen to be *non-projective*. Intuitively, this means that in a given dependency tree there exist arcs that cross each other. As highlighted in the example from Figure 2, a non-projective arc can be detected due to the presence of two crossing arcs: from the token *saw* to *yesterday* and from the token *person* to *dog*.

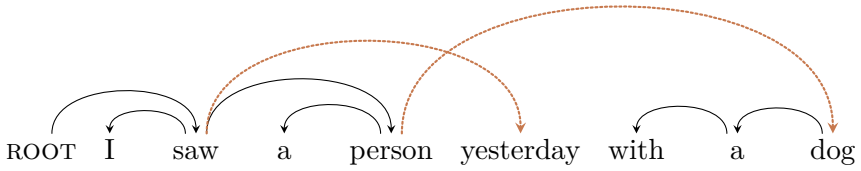


Figure 2: An example of a non-projective dependency tree.

More formally, non-projectivity can be determined as follows:

- NON-PROJECTIVITY if there are arcs $w_h \rightarrow w_d$ and $w_{h'} \rightarrow w_{d'}$: $min(h, d) < min(h', d') < max(h, d) < max(h', d')$

Handling non-projectivity in dependency parsing has been shown to be challenging. As some previous analyses have demonstrated (McDonald and Satta, 2007; Gómez-Rodríguez, 2016), solving non-projectivity often results in computationally costly algorithms. Since the degree of non-projectivity varies across languages, some parsers are therefore designed to only support projective trees in order to preserve the efficiency.

To overcome this shortcoming, one may apply a property of *k-planarity* that provides an efficient solution to deal with non-projectivity and that we will cover in Section 4.1.2.1.

2.2 Dependency treebanks

The emergence of corpora with dependency annotation has been crucial for training and testing data-driven parsers on a range of languages. They also serve for challenging parser's capabilities regarding some specific aspects, for instance, to improve multilingual or low-resource parsing[1]. Furthermore, some dependency treebanks are enhanced with additional annotations coming from e.g. other parsing paradigms (Seddah et al., 2013) or eye-tracking data (Barrett, Agić, and Søgaard, 2015) that may provide complementary representations useful for a dependency parser.

In this section, we will outline the common format of dependency treebanks and we will describe the selected corpora, as well as, the evaluation metrics used in our work.

2.2.1 CoNNL-X format

Previously, disparity in annotation schemes, evaluation methods or data splits was posing challenges for a direct comparison of dependency parsers (Buchholz and Marsi, 2006). Hence, considerable effort has been put into standardizing the dependency format across languages giving origin to the CoNNL-X format (Buchholz and Marsi, 2006). In general, a CoNNL-X file is a text file containing sentences separated by an empty line and where each token in a sentence consists of 10 fields delimited by tabs. In recent years, a modified version of it has been commonly used: the CoNNL-U format that is annotated according to the Universal Dependencies guidelines (Nivre et al., 2016). Figure 3 shows an exemplary snippet of an annotated sentence in English.

```
# sent_id = weblog-blogspot.com_marketview_20040611132900_ENG_20040611_132900-0009
# text = The other problem?
1  The      the      DET    DT  Definite=Def|PronType=Art  3  det    3:det    _
2  other    other    ADJ    JJ  Degree=Pos                 3  amod   3:amod   _
3  problem  problem  NOUN   NN  Number=Sing                0  root   0:root   SpaceAfter=No
4  ?        ?        PUNCT  .   _                          3  punct  3:punct  _
```

Figure 3: An excerpt in CoNNL-U format from the $\text{English}_{\text{EWT}}$ UD treebank.

Each sentence in the CoNNL-U format[2] is preceded with additional annotation marked with a hash sign, followed by lines corresponding to

1 An overview of previous CoNLL Shared Tasks on dependency parsing is available at `https://www.conll.org/previous-tasks`

2 https://universaldependencies.org/format.html

each word in a given sentence. Such a line starts with a column denoting the word index (ID) in the sentence, counting from 1, its form (FORM) and lemma (LEMMA). Then two fields with Part-of-Speech (PoS) tag information are provided: UPOS and XPOS, where the former stands for the Universal PoS tags, while the latter for language-specific ones. Next column corresponds to morphological features (FEATS), followed by a column containing the index of the word's head (HEAD). If HEAD is 0, then the head for that word is the artificial ROOT of the sentence. Next, there is a field describing the dependency relation type (DEPREL), while the last columns refer to the head-deprel pairs (DEPS) and additional annotations (MISC). If for a given field the information is unavailable, then the field is substituted with an underscore.

2.2.2 English Penn Treebank (PTB)

The English Penn Treebank (Marcus, Marcinkiewicz, and Santorini, 1993) is a licensed English treebank containing a section based on the Wall Street Journal (WSJ) articles that is commonly used in syntactic parsing. Since this treebank was annotated according to the constituency grammar, it requires a conversion to dependencies using some external tools. Various consecutive SOTA dependency parsers (Chen and Manning, 2014; Dyer et al., 2015; Kiperwasser and Goldberg, 2016; Ma et al., 2018; Fernández-González and Gómez-Rodríguez, 2019) use the following standard splits: sections 2–21 for training, 23 as the development set, and 24 as the test set, and rely on the Stanford Dependency (SD) conversion (Marneffe, MacCartney, and Manning, 2006) and the Stanford PoS tagger (Toutanova et al., 2003). We follow this setup in our experiments in order to provide a comparison of our models against some existing dependency parsers. It is also worth mentioning that a common practice is to exclude punctuation from the evaluation when using this PTB-SD treebank.

2.2.3 Universal Dependencies (UD)

Universal Dependencies (UD) (Nivre et al., 2016) is an open project addressing the need for multilingual treebanks that are based on a coherent annotation scheme across the languages. It is a dynamically growing project and nowadays, the latest released version (Zeman et al., 2021) provides a collection of 202 treebanks in 114 languages[3], enabling to redirect the focus on parsing advances to other languages than English.

Apart from UD being a useful resource with dependency annotation across languages, it is also used for developing and testing multilingual parsers in more real-world settings. For instance, CoNLL Shared Tasks

3 Checked on the 1st of July 2021.

2017[4] and 2018[5] aimed at improving parsing from a raw text to universal dependencies and resulted in various systems that could be directly compared on a wide range of UD languages (Kübler, McDonald, and Nivre, 2009).

In our work, we evaluate the models on a subset of treebanks either from UDv2.2 (Nivre et al., 2018) or UDv2.4 (Nivre et al., 2019). To ensure an evaluation on a representative group of languages, we follow the criteria for a treebank choice (i.a. typology, treebank size, degree of non-projectivity, morphological richness) proposed by Lhoneux, Stymne, and Nivre (2017b) and Anderson and Gómez-Rodríguez (2020a). In the case of experiments on non-projectivity (see Section 4.3.2), we extract UD treebanks with the highest percentage of non-projective sentences. Moreover, in order to liken a more realistic setup as in CoNLL Shared Tasks, we use UDPipe models (Straka and Straková, 2017) to obtain the predicted segmentation and tokenization for the selected treebanks. It is also worth mentioning that it is a standard practice to include punctuation when evaluating a parser on the UD treebanks.

2.2.4 Evaluation metrics

There are several evaluation metrics to measure the performance of a dependency parser. The most commonly used are the following:

- Unlabeled Attachment Score (UAS): Percentage of correctly assigned heads.
- Labeled Attachment Score (LAS): Percentage of correctly assigned heads and dependency relations.

Additionally, in the experiments where we examine the improvements in the coverage of non-projective arcs, we evaluate the parser's performance using:

- Precision (P): Percentage of correct non-projective arcs out of all predicted non-projective arcs.
- Recall (R): Percentage of correct non-projective arcs out of all gold non-projective arcs.

2.3 A shift towards neural dependency parsing

Traditionally, there have been two prevalent approaches in data-driven dependency parsing: transition-based (Yamada and Matsumoto, 2003; Covington, 2001; Nivre, 2003) and graph-based (Eisner, 1996;

4 CoNLL 2017 Shared Task: http://universaldependencies.org/conll17/
5 CoNLL 2018 Shared Task: http://universaldependencies.org/conll18/

McDonald, Crammer, and Pereira, 2005). Generally, these canonical approaches differ in how a dependency tree is inferred (McDonald and Nivre, 2011). Namely, in transition-based systems a dependency tree is greedily constructed based on a sequence of transitions, where for each time step given some configuration the most optimal transition is opted. In the graph-based approach, in turn, the highest scored dependency tree based on the score of its arcs is globally inferred from the set of all possible trees.[6]

Recent years have seen a surge of interest in neural networks that have provided many advances and SOTA results for a range of NLP tasks. Likewise, transition- and graph-based parsers that formerly were often based on linear models have been enhanced with the neural network methods. One of the motivations for the shift to neural dependency parsing is that these methods implicitly perform an efficient feature extraction that used to be the bottleneck of the pre-neural dependency parsers (particularly in transition-based models).

2.3.1 Feature representations

Parser's decisions are made based on a feature model. For that reason it is crucial to provide a model with the most useful feature representations. In the pre-neural transition- and graph-based parsing, models used to rely on some predefined feature templates (Zhang and Clark, 2008; Zhang and Nivre, 2011). However, as Chen and Manning (2014) notice such feature representations can be sparse and incomplete, while their retrieval is also computationally demanding (Vilares and Gómez-Rodríguez, 2018). With the rise of neural networks methods this could be overcome by replacing sparse features with their dense representations and by introducing non-linear feature extractors. Particularly, Chen and Manning (2014) show that their model based on Multi-Layer Perceptron (MLP) can successfully learn from a small set of atomic features represented as dense vectors, mitigating the need of defining a large amount of hand-crafted feature representations beforehand. Subsequently, Kiperwasser and Goldberg (2016) show that when substituting MLP with a BiLSTM architecture, the number of features could be reduced even further and eventually, Shi, Huang, and Lee (2017) propose a minimal set of two features that suffices to obtain good parsing performance.

When it comes to graph-based parsing, a wide range of pre-neural parsers used to primarily rely on first-order modeling that only considers single arcs, unlike higher-order models that make use of richer features spanning to grandparent or sibling arcs. The reason for that is that first-order parsers used to be considerably more efficient even when handling non-projectivity, however with the limitation of relying

6 A more detailed description of both approaches can be found in Kübler, McDonald, and Nivre (2009).

on solely local features. The higher-order parsers, in turn, used to provide further improvements in accuracy but easily become intractable with arbitrary non-projectivity (McDonald and Satta, 2007). However, it has been shown that higher-order parsing is indeed tractable for projective parsing (Koo and Collins, 2010) and even may support non-projectivity and be fast when using an approximate inference (Martins, Almeida, and Smith, 2013). With the shift to neural parsers, Pei, Ge, and Chang (2015) demonstrate that employing MLP eases learning of high-order feature combinations. However, it has been later shown that BiLSTM-based models that rely on the first-order features suffice to obtain a competitive performance (Kiperwasser and Goldberg, 2016) and even SOTA results when extended with a biaffine attention (Dozat and Manning, 2017). Newly, it has been shown that with GPU parallelisation capabilities and batching methods, higher-order modeling can become even more efficient (Zhang, Li, and Zhang, 2020).

2.3.2 Pre-trained and deep contextualized word representations

With the introduction of neural networks, words became represented as low-dimensional dense vectors called word embeddings. Nowadays, neural parsers often rely on word embeddings that were pre-trained on some external corpora. They are easily pluggable into a parser and endow it with some initialized word representations leading to performance improvements. The earliest pre-trained word embeddings, such as: word2vec (Mikolov et al., 2013), GloVe (Pennington, Socher, and Manning, 2014) or fastText (Bojanowski et al., 2017) are however static and do not reflect the word order. Hence, other embeddings have been developed to be more suitable for the syntax-oriented tasks. For instance, Ling et al. (2015) present structured word embeddings considering the word order, while Levy and Goldberg (2014) suggest dependency-based embeddings that define context with respect to the syntactic dependencies. As a side note, the common use of the pre-trained word embeddings in parsing has raised a question, to what extent a parser should be perceived as fully- or semi-supervised, since the pre-trained word embeddings provide a parser with some external knowledge. A reader may see remarks on it in the work of Kiperwasser and Goldberg (2016) and Fernández-González and Gómez-Rodríguez (2019).

Lately, the deep contextualized word representations have gained much attention. This type of word representations comes from pre-trained language models, such as ELMo (Peters et al., 2018) based on LSTM (Hochreiter and Schmidhuber, 1997) or BERT (Devlin et al., 2019) enhanced with transformers (Vaswani et al., 2017). The deep contextualized embeddings are more powerful than the static ones, since their representations are generated based on the sentential con-

text in which a given word appears. However, in order to use them a parser needs to embed the pre-trained model that often considerably increases the parser's parameter space and complexity. Recently, syntactic parsers augmented with such models have achieved SOTA results in both constituency (Kitaev and Klein, 2018; Zhang, Zhou, and Li, 2020) and dependency parsing (Zhou and Zhao, 2019; Mrini et al., 2020) generating interest in understanding what structural representations the pre-trained models acquire (Hewitt and Manning, 2019; Tenney et al., 2019). Interestingly, Vilares et al. (2020) show that syntactic parsing can even be reduced to pre-training by only relying on the vectors from the pre-trained encoders demonstrating that this kind of models are endowed to some extent with the structural awareness.

2.3.3 Selected neural network architectures

Now we will describe some architectures based on the recurrent neural networks that are relevant for our work and their previous application in dependency parsing.[7]

RNN Recurrent Neural Networks (Elman, 1990) are descendants of the feed-forward networks with the ability to encode the input sequentially. More concretely, RNN enables processing of each element in the sequence conditioning on the previously processed input. Hence, a hidden layer at a time step t is a function of the weighted hidden layer from the previous time step, the weighted input from the current time step and the bias term which are fed to a non-linear function σ. It can be defined as:

$$h_t = \sigma(Uh_{t-1} + Wx_t + b) \tag{1}$$

However, there are some limitations for this kind of recursive architecture. Since the input sequence may be of an arbitrary length, larger input sequences may be harder to learn and some information may get lost. Moreover, it may lead to problems of vanishing or exploding gradients (Pascanu, Mikolov, and Bengio, 2013). In order to tackle this issue, a variant of RNN has been proposed that we will describe next.

LSTM Long Short-Term Memory networks (Hochreiter and Schmidhuber, 1997) are enhanced with gates to improve the control of the information that needs to be preserved or added. Concretely, they are endowed with the forget, input and output gates that are defined as:

7 A more detailed description of the following neural architectures can be found in Goldberg (2017) and `https://web.stanford.edu/~jurafsky/slp3/`.

$$f_t = \sigma(U_f h_{t-1} + W_f x_t + b_f) \quad (2)$$
$$i_t = \sigma(U_i h_{t-1} + W_i x_t + b_i) \quad (3)$$
$$o_t = \sigma(U_o h_{t-1} + W_o x_t + b_o) \quad (4)$$

Furthermore, a memory cell c_t is introduced to determine how much information from previous states should be retained and how much new information should be added to the context. Hence, the current hidden layer is then computed based on the output gate and the memory cell passed through a non-linear function. More formally:

$$g_t = tanh(U_g h_{t-1} + W_g x_t + b_g) \quad (5)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \quad (6)$$
$$h_t = o_t \odot tanh(c_t) \quad (7)$$

The LSTM architecture has been shown to be well-suited for the dependency parsing problem. For instance, Dyer et al. (2015) use a stack LSTM to obtain representations of the entire stack and buffer and thereby obviate the need of defining features with respect to the positions of elements on the stack and buffer. Additionally, the authors make use of embeddings for partly constructed subtrees on the stack by applying a composition function from recursive neural networks. Furthermore, Ballesteros et al. (2016) improve the LSTM-based parser by adding a dynamic oracle.

However, processing an input in the left-to-right fashion constrains the context to the information coming from the previously processed elements. Hence, to alleviate this limitation and enhance the context with the information about the entire sequence, a variant of LSTM has been proposed that processes the input in both directions.

BiLSTM Bidirectional LSTMs (Graves and Schmidhuber, 2005) enable to combine left and right context for an input x_i. To do so, one LSTM processes the input from the beginning of a sequence up to position i, while the second one processes the input from the end of a sequence up to position i and the representations from both directions are then concatenated. The input element is thereby enriched with a global context based on the entire sequence. It can be formalised as:[8]

$$BiLSTM(x_{1:n}, i) = LSTM^{\rightarrow}(x_{1:i}) \circ LSTM^{\leftarrow}(x_{n:i}) \quad (8)$$

One of the first parsers that applies BiLSTMs to transition- and graph-based systems is the BIST parser (Kiperwasser and Goldberg, 2016), in which word and PoS tag embeddings for each token are concatenated

8 $\circ$ denotes a concatenation operator.

and then passed through BiLSTM layers. Then with the feature function the selected BiLSTM feature vectors are concatenated and scored using a non-linear function. The strength of using BiLSTM-based representations of a token is that they are enhanced with a sentential context.

In the transition-based variant of the BIST parser, four positional features are used to score each transition from an arc-hybrid system: the three top elements on the stack and the top element in the buffer. However, Shi, Huang, and Lee (2017) show later that even two positional BiLSTM features suffice. An example of a successor of the transition-based BIST parser is UUParser (Lhoneux et al., 2017) that i.a. relies on character-based embeddings instead of PoS tags and uses a SWAP transition to support non-projectivity (Lhoneux, Stymne, and Nivre, 2017a).

The graph-based variant of the BIST parser is an arc-factored model.[9] BIST parser scores each arc represented as concatenated BiLSTM vectors of a head and modifier using MLP. Dozat and Manning (2017) enhance this approach by introducing a biaffine attention mechanism as a scoring function of arcs and labels resulting in a SOTA parser. Recently, Zhang, Li, and Zhang (2020) extend the biaffine parser with second-order features showing that a higher-order modeling that earlier was challenging could be remedied with an efficient batchifying method. Moreover, the proposed parser yields improvements in accuracy due to the explicitly encoded second-order features suggesting that this type of features is not captured implicitly by BiLSTMs in contrast to what was suggested earlier.

This, in turn, relates to the question of what structural representations are implicitly encoded by BiLSTMs. Falenska and Kuhn (2019) demonstrate that BiLSTM-based parsers are able, in fact, to contain information about some complex syntactic relations. As a result, this architecture alleviates the need of explicitly defining some syntactic features since they are already captured by BiLSTMs. Moreover, their ablation study confirms that the structural context that is encoded by BiLSTMs implicitly has a direct translation into an improved performance of a parser.

BERT Bidirectional Encoder Representations from Transformers (Devlin et al., 2019) is a language model containing deep contextualized representations by relying on transformers (Vaswani et al., 2017). The model is pre-trained on two unsupervised tasks: next sentence prediction and masked language modelling, where the latter serves to provide the model with a bidirectional awareness of the context. BERT is usually used together with a downstream task according to which the parameters are fine-tuned.

9 Around the same time, an alternative BiLSTM-enhanced graph-based parser was proposed by Wang and Chang (2016).

Recently, BERT has become a backbone of many NLP models. Dependency parsers augmented with BERT representations often yield SOTA results (Zhou and Zhao, 2019) and also has enabled multilingual parsing with a single model (Kondratyuk and Straka, 2019). After a range of successful applications of BERT, researchers have addressed the question of BERT's syntax awareness (Goldberg, 2019; Hewitt and Manning, 2019; Tenney et al., 2019). For instance, Jawahar, Sagot, and Seddah (2019) investigate how BERT encodes syntactic representation in English and observe that the phrasal information is captured in the lower layers, more complex syntactic features in the middle layers while deeper layers are needed in order to handle long-distance dependencies.

2.3.4 Non-neural versus neural dependency parsers

The pre-neural transition- and graph-based parsers, such as MaltParser (Nivre et al., 2007) and MSTParser (McDonald, Crammer, and Pereira, 2005) used to provide a similar accuracy even though they originate from different paradigms. The analysis of McDonald and Nivre (2011) showed that one of the main theoretical distinctions between the two parsers is their inference method and the feature representations they rely on. Namely, the transition-based parsers exploit much richer representations based on the entire transition history, while the graph-based ones solely rely on local features restricted to a single arc (unless the latter includes more expensive higher-order feature modeling). As a consequence, different error types emerge in each system. For instance, a transition-based parser is more prone to suffer from error propagation. The reason for that is that it may make a non-optimal decision due to the reliance on local context, and such a decision may cause future errors, unlike the graph-based parser that relies on a global inference. They also have different strengths and weaknesses i.a. the accuracy of transition-based parsers is higher on short dependency arcs and sentences, while the graph-based parsers score better on long ones.

Recently, dependency parsers have been advanced with neural network methods which simplify the feature extraction methods and provide SOTA results. Hence, the canonical approaches became a subject of reexamination in the context of the neural enhancements. For instance, Lhoneux, Stymne, and Nivre (2017b) provide a comparison of non- and neural transition-based parsers using the same transition system algorithm and show that a neural parser improves accuracy on long dependencies while the non-neural counterpart still performs slightly better on short dependencies.

Kulmizev et al. (2019), in turn, investigate the impact of deep contextualized word embeddings on the error types that the transition- and graph-based parsers commit with a close resemblance to the analysis done by McDonald and Nivre (2011). They compare the performance of the extended BiLSTM transition- and graph-based parsers (Kiper-

wasser and Goldberg, 2016; Lhoneux et al., 2017) against their variants enhanced with deep contextualized embeddings, such as ELMo and BERT. The study shows that in general the error patterns for the graph-based and transition-based parsers relying on the BiLSTM architecture remain similar to the ones present in the non-neural counterparts. However, the BiLSTM graph-based model narrows the accuracy gap in short dependencies and sentences with respect to the transition-based parser which, in turn, suffers less from error propagation. Regarding transition- and graph-based models that additionally use ELMo and BERT, it is discernible that their performance becomes even more similar. The authors conclude that the deep contextualized word embeddings enable both parsers to bridge the gap with respect to the accuracy and the types of errors. In particular, the transition-based model makes better local decisions and obtains improvements in long dependencies, while both transition- and graph-based parsers improve on long sentences.

Previously, combining strengths of the pre-neural transition- and graph-based parsers has been shown to be beneficial (Nivre and McDonald, 2008; Zhang and Clark, 2008) since it facilitates finding a trade-off between rich feature representations and exhaustive search. Nevertheless, with the emergence of BiLSTMs and pre-trained models, a distinct error pattern for each approach got blurry. In this context, Falenska, Björkelund, and Kuhn (2020) examine whether integrating a transition- and graph-based parser is useful when relying on BiLSTM representations and deep contextualized embeddings. In general, the authors conclude that the advantage of combining the two approaches no longer is reflected in the accuracy.

As an outcome of applying neural methods to dependency parsing and more efficient feature extraction methods, the traditional division between transition- and graph-based parsers has faded. It has also become prominent that the parsing advances have been driven to a larger extent by applying various neural architectures than by the parsing algorithms themselves. Hence, this enables redirecting the focus on other aspects of dependency parsing, such as architectural simplicity, the capability of easily integrating dependency parser in other systems or parsing speed.

2.4 Parsing speed breakdown

The past decade has seen great improvements in dependency parsing accuracy, however the performance of a parser can seem deceiving when merely measured in terms of its precision while disregarding its computational efficiency. Dependency parsers vary with respect to the trade-off between the speed and accuracy they provide. Therefore, some of them, even if capable of obtaining SOTA results, may not be apt for large-scale parsing (Gómez-Rodríguez, 2017) or integration in other

Parser	Reported by	Sent/s		UAS
		CPU	GPU	
Chen and Manning (2014)	Chen and Manning (2014)	654	–	91.80
Ballesteros, Dyer, and Smith (2015)	Gómez-Rodríguez (2018)	22.7	–	91.63
Kiperwasser and Goldberg (2016) (T)	Strzyz et al. (2019c)	76	–	93.90
Kiperwasser and Goldberg (2016) (G)	Strzyz et al. (2019c)	80	–	93.10
Kuncoro et al. (2016)	Kuncoro et al. (2016)	20	–	94.26
Dozat and Manning (2017)	Dozat and Manning (2017)	–	411	95.74
Smith et al. (2018)	Anderson and Gómez-Rodríguez (2020a)	42	–	94.63
Ma et al. (2018)	Fernández-G and Gómez-R (2019)	–	10.24	95.87
Fernández-G and Gómez-R (2019)	Fernández-G and Gómez-R (2019)	–	23.8	96.04
Zhou and Zhao (2019)	Zhou and Zhao (2019)	–	158.7	96.09
Anderson and Gómez-Rodríguez (2020a)	Anderson and Gómez-Rodríguez (2020a)	96	1153	94.59
Zhang, Li, and Zhang (2020)	Zhang, Li, and Zhang (2020)	–	400	96.11

Table 1: Overview of parsing speeds for some neural dependency parsers evaluated on the PTB-SD test set.

NLP systems. In the context of the latter, Gómez-Rodríguez, Alonso-Alonso, and Vilares (2019) investigate the impact of parsing speed and accuracy on sentiment analysis as the end task and argue that more importance should be attached to parsing efficiency. The reason for that is that small differences in accuracy between parsers may, in fact, not be discernible in the performance of the end task, while parsers' inefficiency has an adverse impact on the entire system.

Furthermore, the need for computationally efficient systems becomes even more prominent when relying on deep neural networks. Neural models often require a large parameter space and costly resources for training. Thus recently, green NLP is generating considerable attention (Strubell, Ganesh, and McCallum, 2019) and great efforts are undertaken in order to provide accurate but smaller and faster models (Jiao et al., 2020; Wang et al., 2020).

In Table 1, we provide an overview of some recent neural dependency parsers with their inference time (measured on CPU and/or GPU), as well as the accuracy (UAS). The list is restricted to the parsers whose parsing speed has been reported. It is also worth noting that this overview only serves for informative purposes since the speeds are often measured on different machines and thus are not directly comparable. Nonetheless, to get some notion about the efficiency of neural parsers, we select ones that are evaluated on the PTB-SD treebank.

Earlier, some attempts have been made with the purpose of reducing parsing speed. For instance, Strzyz and Gómez-Rodríguez (2019) provide some heuristics for reusing partial results automatically from previously parsed tree substructures, while Vieira and Eisner (2017) propose a pruning method in order to preserve the speed-accuracy balance in constituent parsing. Regarding neural dependency parsing, one of the efficient methods for decreasing the inference time is a teacher-student distillation technique, where the knowledge from a larger model is injected into a smaller and faster one (Anderson and Gómez-Rodríguez,

2020a). Furthermore, Strubell and McCallum (2017) suggest a novel architecture to better leverage the GPU power in dependency parsing.

In this thesis, we propose an approach, where dependency parsing is recast as sequence labeling task as an attempt of providing a parser that maintains a good speed-accuracy equilibrium. Our method is promising since it does not require any traditional parsing algorithms or auxiliary structures. Additionally, it can be easily integrated with other downstream tasks. Hence, now we turn to formulate the sequence labeling problem.

2.5 Sequence labeling

Sequence labeling is a structured prediction task, where for each input token a single output label is generated. Hence, an input sentence $w = w_1, w_2, ..., w_n$ is assigned a sequence of labels $[l_1, l_2, ..., l_n]$ of the same length coming from a discrete set of labels L.

Traditionally, sequence labeling tasks, such as: PoS tagging, Named Entity Recognition (NER) or chunking used to rely on pre-neural machine learning models like Hidden Markov Models (HMM) or Conditional Random Fields (CRF) (Lafferty, McCallum, and Pereira, 2001). With the rise of neural network methods, sequence labeling models have been enhanced with i.a. Convolutional Neural Networks (CNN) (Collobert et al., 2011), BiLSTM (Huang, Xu, and Yu, 2015; Ma and Hovy, 2016; Lample et al., 2016) and Gated Recurrent Units (GRU) (Yang, Salakhutdinov, and Cohen, 2017) mitigating the need of hand-crafted feature engineering.

Figure 4 illustrates an exemplary neural architecture for sequence labeling with the objective of predicting a PoS tag for each input token. As shown in the example, the lowest layers may serve for obtaining character representations for the input token using recurrent or convolutional neural networks. Furthermore, these representations can be learned together with word representations and any other selected feature. A sequence of output labels may be then generated based on CRF or a *softmax* in the top layer. An advantage of using CRF as an inference method is that, unlike *softmax*, the dependencies between the output labels are taken into account. The latter however renders a more efficient inference (Yang and Zhang, 2018) but often results in lower accuracy.

Since some of the sequence labeling tasks are to some extent related, it has been shown that a model can benefit from sharing their representations during training (Collobert et al., 2011; Yang, Salakhutdinov, and Cohen, 2016). It is an example of Multi-task learning (MTL) that we will discuss in the following section.

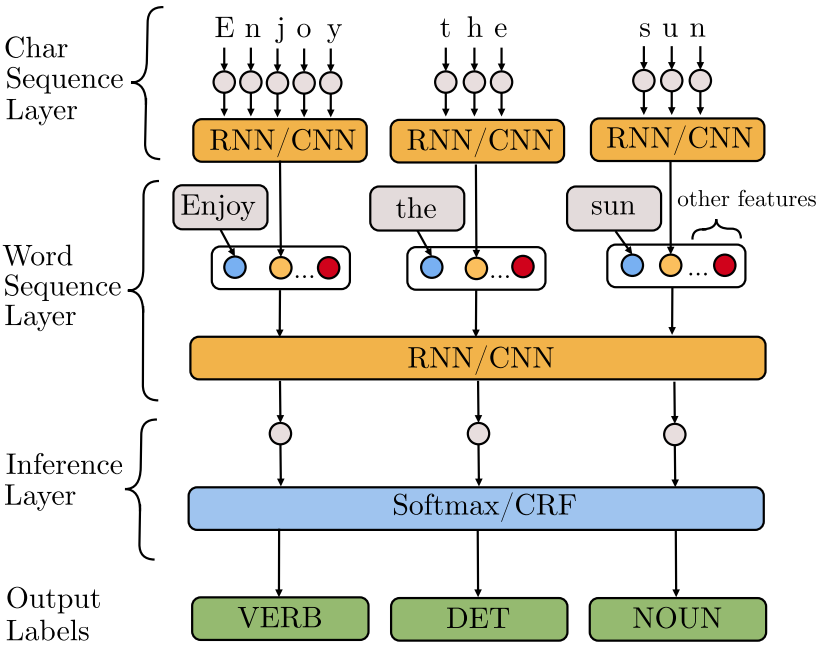


Figure 4: An example of a sequence labeling architecture for PoS tagging. The image is adapted from Yang and Zhang (2018) for the NCRF++ toolkit.

2.6 Multi-task learning

The concept of Multi-task learning (MTL) refers to a method for joint learning of related tasks, where a model leverages the shared representations and thereby improves its generalization abilities (Caruana, 1997). In deep neural networks it is common to deploy it with *hard parameter sharing*, where the hidden layers are shared across the model while keeping separate layers for computing output for each task respectively (Ruder, 2017). This kind of architecture is illustrated in Figure 5.

Moreover, tasks may be defined as *main* or *auxiliary* in the MTL setup. The distinction between them is that auxiliary tasks serve to improve the main task by sharing the representations but the actual output of such additional tasks is disregarded. In this context, some work has called into question which tasks are most suitable for a joint training and how to maximize the leverage of such representations (Martínez Alonso and Plank, 2017; Bingel and Søgaard, 2017; Changpinyo, Hu, and Sha, 2018; Schröder and Biemann, 2020).

MTL has lately received much attention and is rife among NLP tasks. In the context of sequence labeling, use of MTL in PoS tagging, NER or chunking has been investigated (Søgaard and Goldberg, 2016; Plank, Søgaard, and Goldberg, 2016; Liu, Winata, and Fung, 2020). Some work also suggests that a syntactic parser may also leverage joint learning with other taks (Coavoux and Crabbé, 2017; Kankanampati et al., 2020).

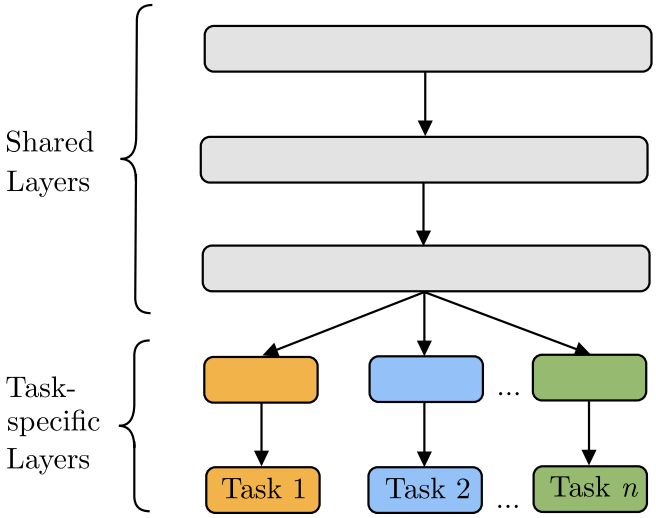
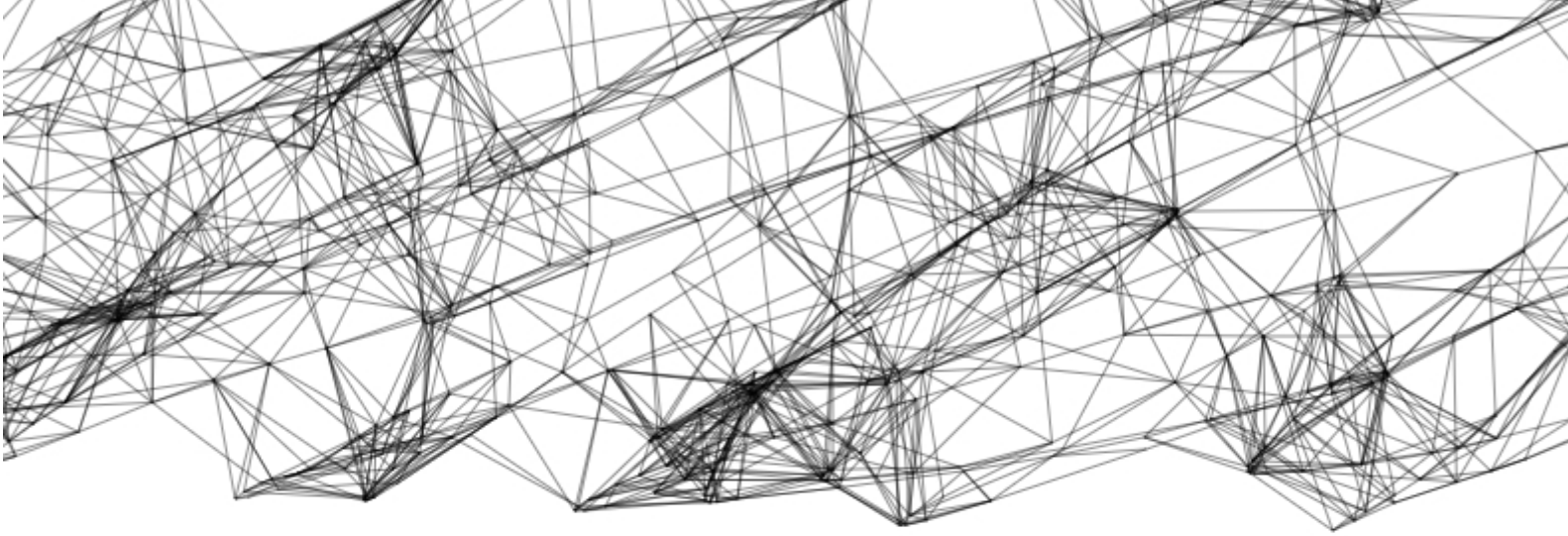


Figure 5: Multi-task learning with hard parameter sharing. The image is adapted from Ruder (2017).

In this thesis, we will apply MTL to dependency parsing as sequence labeling in order to examine whether it renders further improvements in our models. The motivation behind using MTL in our approach is two-fold. Firstly, MTL enables decomposing a label into its atomic elements and learning them as separate main tasks. An upshot of this is that it reduces the label sparsity facilitating learnability of label distributions and it may mitigate the problem of out-of-vocabulary labels at the test time. Furthermore, the inference time may decrease since instead of using a single large *softmax* for all labels, the model will contain several smaller *softmaxes* corresponding to each atomic label. Secondly, MTL facilitates widening our approach by adding auxiliary tasks from complementary external data. Particularly, we will use this setup to learn constituency parsing labels and labels retrieved from eye-tracking data.



Part II

ENCODINGS

The backbone of recasting dependency parsing as sequence labeling are encodings, which are linearization methods for representing a dependency tree as a sequence of labels (one per word). This part describes three encoding families: (i) head selection, (ii) bracketing-based and (iii) transition-based encodings, which differ in how label representations of a dependency tree are retrieved.

3 Head selection encodings

The main trait of this encoding family is that the position of the word's head is encoded in the label for that word either directly or in a relative way that can depend on other linguistic elements in a sentence. More concretely, we will introduce three encodings belonging to this family that use the absolute position of the head, relative distance from the token to the head or the distance with respect to the head's PoS tag. In more concrete terms, a label l_i for a word w_i can be generalized to the form of a tuple $l_i = (e_i, r_i)$, where e_i encodes the head position, while r_i the dependency relation type for the arc $w_j \xrightarrow{r_i} w_i$.

3.1 Encodings

3.1.1 Naive positional encoding

In the naive positional encoding, e_i for a word w_i stores the absolute position of its head in the sentence. Hence, the position of the head w_j at index j of a word w_i at index i is encoded as $x_i = j$ with the corresponding dependency type for that arc $w_j \xrightarrow{r_i} w_i$. An example of this encoding is illustrated in Figure 6.

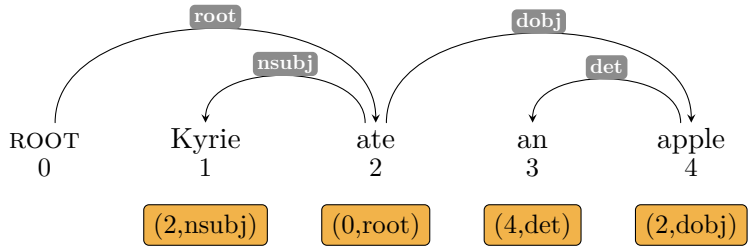


Figure 6: A dependency tree represented with the naive positional encoding.

3.1.2 Relative positional encoding

In the relative positional encoding, in turn, e_i captures the positional difference between the index of the head w_j and its dependent w_i. Hence, the head position is defined as j such that $x_i = j - i$ and the dependency type $w_{x_i+i} \xrightarrow{r_i} w_i$. An example of such a label assignment is shown in Figure 7. This encoding method has been used in some previous work (Li et al., 2018; Kiperwasser and Ballesteros, 2018).

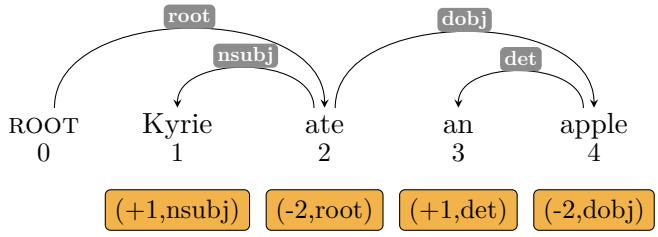


Figure 7: A dependency tree represented with the relative positional encoding.

3.1.3 Relative PoS-based encoding

In the relative Part-of-Speech-based encoding, e_i is represented as a tuple (o_i, p_i) that stores the positional distance (o_i) between the head and the dependent with respect to some particular PoS tag (p_i). Hence, if $o_i > 0$ the head of w_i is o_i positions to the right considering only words with the specific PoS tag p_i. Conversely, if $o_i < 0$ the head of w_i is o_i positions to the left counting words with a PoS tag p_i. The corresponding dependency type is assigned for that arc, such that $w_{(o_i,p_i)} \xrightarrow{r_i} w_i$. The positional distance to the ROOT is encoded as (-1, ROOT) assuming that its index position in a sentence is 0. An example of this encoding is shown in Figure 8 where, for instance, the token $Kyrie_1$ is assigned a label (+1,V) denoting that its head is the first word to the right (+1) with the PoS tag V. This method is similar to the one proposed by Spoustová and Spousta (2010).

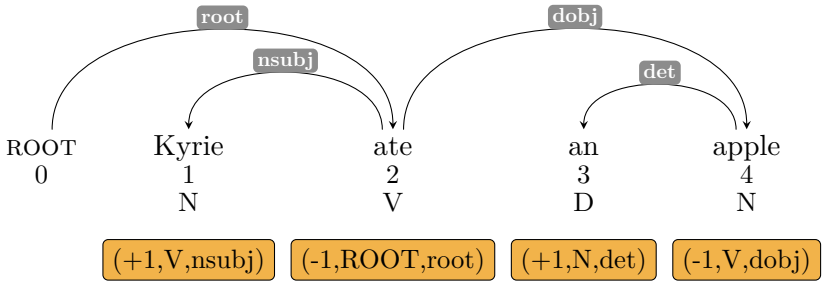


Figure 8: A dependency tree represented with the relative PoS-based encoding.

3.2 Decoding

Following Gómez-Rodríguez and Vilares (2018), our encodings can be shown to be *complete* since they are capable of representing any arbitrary tree as a sequence of labels. They are also *injective* because each sequence of labels corresponds to only a single dependency tree. However, they are not *surjective*, since some labels in the predicted label sequence may be incorrect and preclude building a dependency tree. In

order to recover from such erroneous labels, some postprocessing steps are needed.

In general, the objective of the decoding process is to build a well-formed dependency tree from a label sequence predicted by a classifier. However, the predicted head position for a given token may fall outside the sentence scope or as in the case of the relative PoS-based encoding, the predicted PoS tag for a certain head may be not present. This impedes locating the correct head token and thus requires applying some postprocessing heuristics.

3.2.1 Postprocessing heuristics

We propose a single postprocessing method for all encodings that guarantees that the output dependency tree is well-formed i.e. each token is associated with a single head, where one of them is attached to the ROOT[1]. In addition, the acyclicity constraint is imposed.

More precisely, we first ensure that there is a token depending on ROOT (in other words, there is a token in a sentence with a head at index 0). If this is not the case, we search for all tokens with the predicted dependency relation "root", since it is a good indicator of potential dependents of *root*. If candidate tokens are found, we choose the first one among them, otherwise we select the first token in the sentence as a dependent of ROOT. All remaining tokens without associated heads are then forced to be dependent on that token (this can happen for the reasons mentioned above, e.g. because the predicted head position falls outside the sentence scope). Finally, we eliminate all cycles in the resulting tree by attaching the leftmost token involved in the cycle to the token dependent on ROOT.

3.3 Models and experiments

In this section, we will empirically test each encoding belonging to the head selection family. In the first experiment, we will examine the accuracy they yield compared to some previous work. Moreover, we will analyse the peculiarities they exhibit. In the second experiment, we will explore the effect of decomposing labels into atomic components and learning them by the means of MTL.

3.3.1 Performance comparison of the head selection encodings

We will now compare the encodings in terms of UAS and LAS scores. Thereafter, we will examine the label set sizes that the encodings gen-

1 We follow the UD framework that requires that every tree has a single ROOT node: https://universaldependencies.org/u/dep/root.html.

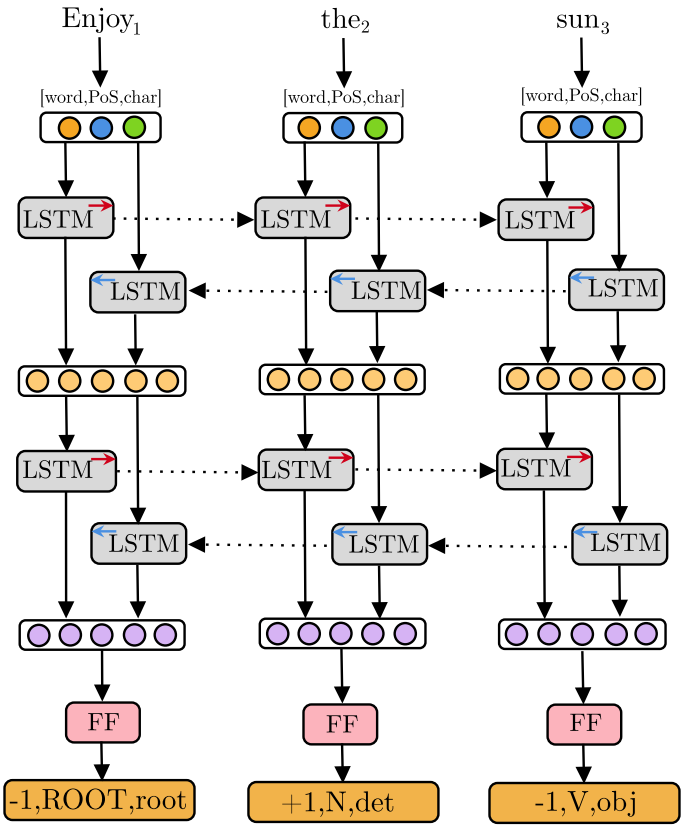


Figure 9: The baseline architecture used in this work for recasting dependency parsing as sequence labeling.

erate and their distributions. Finally, we will explore the impact of using universal and more fine-grained PoS tags on the accuracy of the relative PoS-based model.

Model Throughout the experiments, we choose NCRF++ as our sequence labeling framework (Yang and Zhang, 2018). We model our task using BiLSTMs (see Section 2.3.3) and we follow the configurations used in the primary work on constituency parsing as sequence labeling (Gómez-Rodríguez and Vilares, 2018). The model consists of two stacked BiLSTMs layers, where the hidden representations h_i from the last BiLSTM layer are passed through a feed-forward layer and each output label is obtained by applying a *softmax* function. The input token is a concatenation of pre-trained word embeddings, character embeddings (learned with a char-LSTM layer trained together with the rest of the network) and PoS tag embeddings.

We optimize the model by minimizing the categorical cross-entropy loss, i.e. $\mathcal{L} = -\sum log(P(y_i|h_i))$. The baseline architecture is illustrated in Figure 9, which we will mainly follow in the succeeding experiments. Any deviation will be explicitly specified. The hyperparameters used in this experiment are detailed in Section A.1.1.

Setup The experiments are conducted using the English PTB treebank with its common setup (see Section 2.2.2), and following Dyer et al. (2015) we use the English pre-trained word embeddings introduced by Ling et al. (2015). Furthermore, we evaluate the models on a representative subgroup of languages from UDv2.2: Ancient-Greek$_{\text{PROIEL}}$, Chinese$_{\text{GSD}}$, Czech$_{\text{PDT}}$, English$_{\text{EWT}}$, Finnish$_{\text{TDT}}$, Hebrew$_{\text{HTB}}$, Kazakh$_{\text{KTB}}$ and Tamil$_{\text{TTB}}$ (see Section 2.2.3). For the UD treebanks, we use the pre-trained word embeddings from Ginter et al. (2017). In order to test our parser in more real-world settings, we apply UDPipe with the pre-trained models from the CoNLL18 Shared Task (Straka and Straková, 2017) to obtain data with predicted segmentation, tokenization and PoS tags. The reported speeds (sentences/second) were measured on a single core of the CPU (Intel Core i7-7700 CPU 4.2 GHz).

Encoding	UAS	LAS
Li et al. (2018)(sequence labeling)	87.58	83.81
Li et al. (2018) (seq2seq)	89.16	84.99
Li et al. (2018) (seq2seq+beam+subroot)	93.84	**91.86**
naive-positional	45.41	42.65
rel-positional	91.05	88.67
rel-PoS	**93.99**	91.76

Table 2: Comparison of the head selection encodings on the PTB development set.

Accuracy First, we examine the accuracy of the head selection encodings in terms of UAS and LAS evaluated on the English PTB development set. For comparison, we include the results of sequence labeling and seq2seq models proposed by Li et al. (2018) that also encode the head position using the relative positional encoding. The results are shown in Table 2. In general, the relative PoS-based encoding yields the highest score among the head selection encodings and is on par with the best seq2seq model. Although, it is worth mentioning that the latter relies on a more complex architecture and includes both beam and subroot decomposition. The results suggest that our sequence labeling parser in particular with the relative PoS-based encoding renders a competitive accuracy compared to more complex models. Among the head selection encodings, the relative PoS-based encoding doubles the score of the naive positional encoding, while it also outperforms the relative positional encoding by almost 3 points of UAS.

Label set size To gain a deeper understanding of the encodings' facets, we examine the label set size that each encoding generates. In Table 3 we report both the number of unique labels per encoding and

the size of each atomic component obtained with a label decomposition (for instance, the relative PoS-based label can be split into atomic elements, such as o_i, p_i and r_i).

Encoding	#Labels	#Atomic labels		
		o	p	r
naive-positional	1676	89	–	44
rel-positional	872	99	–	44
rel-PoS	995	33	29	44

Table 3: Label set sizes generated by the head selection encodings on the PTB dev set. Each label is split into its atomic components, where o stands for the head positions (integer), p PoS tags and r the dependency relation types. The size of the latter is always equal for all encodings.

In general, it is discernible that the naive positional encoding generates almost 1.7x more labels than the relative PoS-based one, while achieving considerably lower accuracy. A plausible explanation for the accuracy differences is that it may be more natural to express the syntactic dependencies in terms of relative positions rather than with raw indices. More intuitively, it seems more appropriate to encode that, for instance, a determiner is likely to depend on the next word than using the raw position of the head that constantly varies within a sentence. Another factor that may have an impact is the architectural choice. For instance, pointer networks (Vinyals, Fortunato, and Jaitly, 2015) is an architecture that is apt for operating on absolute positions (Ma et al., 2018; Fernández-González and Gómez-Rodríguez, 2019), however due to the contextual nature of BiLSTM, one may presume that a model based on the relative position encoding may produce better results.

Furthermore, when looking at the sizes of atomic labels, the relative PoS-based encoding generates overall the smallest number of labels for expressing the head positions. We will show later how this observation can be leveraged in practical terms with MTL. In Figure 10, we visualize the label distribution of each encoding with respect to its label frequency. The relative positional and relative PoS-based encodings seem to produce a larger number of highly occurring labels, while the naive encoding generates a longer tail.

In sum, the relative PoS-based encoding yields the best results among the head selection encodings. However, one of its peculiarities is that it relies on PoS tags. Hence, we move now to investigate the impact of PoS tag types on this encoding.

Role of Part-of-Speech tags Treebanks in different languages vary in the quantity and quality of the PoS tags they contain. This potentially can influence the performance of the relative PoS-based encoding due do its strong reliance on them. On these grounds, we explore the sensitivity of the encoding with respect to the universal PoS tags

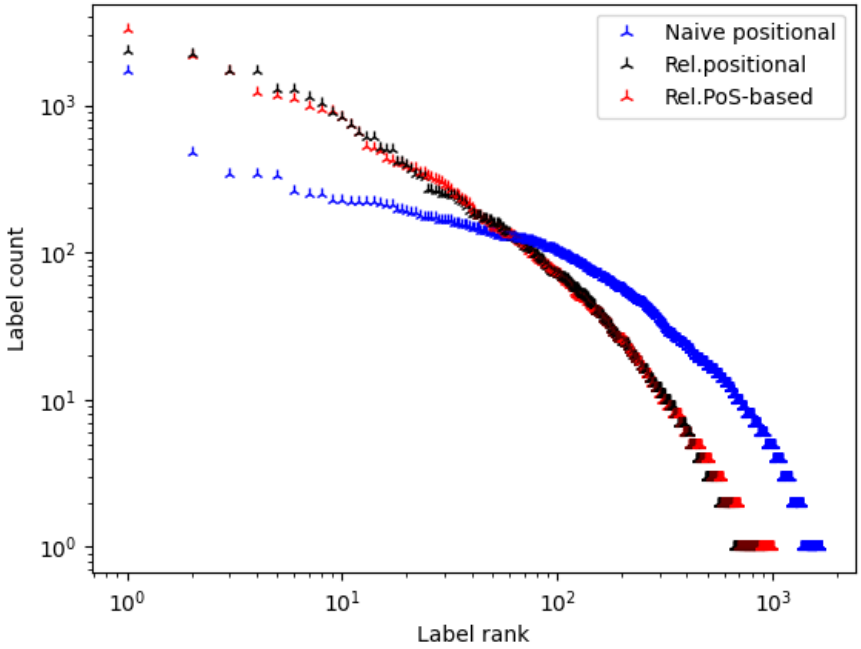


Figure 10: Distribution of ranked labels with respect to their occurrences in the PTB dev set represented in a log-log scale.

(UPoS) and the language-specific ones (XPoS), where the latter is often more fine-grained, however not always provided for a given language of interest.

Table 4 shows the accuracy of models trained with the UPoS- and XPoS-based encoding. In general, the results suggest that models relying on UPoS tags yield better accuracy across the languages by outperforming XPoS-based models on 5 out of 7 treebanks. A plausible reason for it may be that the number of unique UPoS tags is often smaller than XPoS tags. This, in turn, may ensure a more optimal label set size and thereby allow a more efficient learning. In fact, in the case of the Czech and Tamil treebanks, it is discernible that a large XPoS tag set hurts model performance arguably due to the large and sparse label space they generate. For comparison, the model for the Czech language relies on 16 unique UPoS tags and generates 1194 labels, while the XPoS-based counterpart operates on 1075 XPoS tags increasing the label set size more than six times. In contrast, the XPoS-based models for Ancient Greek and Kazakh languages perform slightly better than the counterpart, although the label set size based on the two tag sets does not differ substantially. The Kazakh treebank is also the smallest one, hence it may not be suitable for generalizations.

2 Hebrew treebank contains equal UPoS and XPoS tags.
3 The scores are based on the test since Kazakh lacks a development set.

	UPoS-based				XPoS-based			
Treebank	UAS	LAS	#UPoS	#Labels	UAS	LAS	#XPoS	#Labels
Ancient Greek$_{\text{PROIEL}}$	76.58	71.70	14	514	**77.00**	**72.14**	23	570
Chinese$_{\text{GSD}}$	**61.01**	**57.28**	15	600	60.98	57.14	37	626
Czech$_{\text{PDT}}$	**89.82**	**87.63**	16	1194	88.33	85.46	1075	6928
English$_{\text{EWT}}$	**82.22**	**78.96**	17	717	82.05	78.70	50	983
Finnish$_{\text{TDT}}$	**80.31**	**76.39**	15	682	80.19	76.28	12	658
Hebrew$_{\text{HTB}}$[2]	67.23	62.86	16	450	67.23	62.86	16	450
Kazakh$_{\text{KTB}}$[3]	32.14	17.03	18	408	**32.93**	**17.07**	32	423
Tamil$_{\text{TTB}}$	**73.24**	**66.51**	13	162	59.70	52.57	121	340

Table 4: Performance comparison of models with the relative UPoS- and XPoS-based encoding on the dev set across several UD languages. The number of UPoS/XPoS tags and labels refers to the distinct tags found in the dev set for each language.

Since we aim to examine the encoding in a more real-world setting where PoS tags (as well as previous steps like segmentation) are predicted, Table 5 provides some statistics on the prediction accuracy of UPoS and XPoS tags for the selected languages using UDPipe (Straka and Straková, 2017)[4]. It is notable that the prediction accuracy of PoS tags varies across the languages which may have an impact on the performance of our models.

Treebank	UPoS(%)	XPoS (%)
Ancient Greek$_{\text{PROIEL}}$	95.80	96.00
Chinese$_{\text{GSD}}$	84.00	83.80
Czech$_{\text{PDT}}$	98.30	92.80
English$_{\text{EWT}}$	93.50	92.90
Finnish$_{\text{TDT}}$	94.50	95.70
Hebrew$_{\text{HTB}}$	80.90	80.90
Kazakh$_{\text{KTB}}$	52.00	52.10
Tamil$_{\text{TTB}}$	82.20	77.70

Table 5: UDPipe models' accuracy on UPoS and XPoS prediction for the selected UDv.2.0 languages using raw text.

Next, we compare the accuracy of the PoS-based models against another dependency parser using the predicted data sets. We choose the transition-based BIST parser (Kiperwasser and Goldberg, 2016) for comparison since similarly to our approach, it relies on a BiLSTM-based architecture. Table 6 reports the accuracy and speeds of both parsers using the PoS tag type that yields the best performance for a given language based on Table 4.

4 UDPipe statistics are available at `https://ufal.mff.cuni.cz/udpipe/1/models`

5 Due to lack of pre-trained UDPipe model, the gold data set was used. We did not use any pre-trained word embeddings.

Treebank	PoS type	rel-PoS (sent/s)	rel-PoS UAS	rel-PoS LAS	BIST (transition-based) (sent/s)	BIST (transition-based) UAS	BIST (transition-based) LAS
Ancient Greek$_{\text{PROIEL}}$	XPOS	$123_{\pm 1}$	75.31	70.87	$116_{\pm 4}$	69.43	64.41
Chinese$_{\text{GSD}}$	UPOS	$105_{\pm 0}$	63.20	59.12	$73_{\pm 1}$	64.69	60.45
Czech$_{\text{PDT}}$	UPOS	$125_{\pm 1}$	89.10	86.68	$94_{\pm 3}$	89.25	86.11
English$_{\text{EWT}}$	UPOS	$139_{\pm 1}$	81.48	78.64	$120_{\pm 2}$	82.22	79.00
Finnish$_{\text{TDT}}$	UPOS	$168_{\pm 0}$	80.12	76.22	$127_{\pm 3}$	80.99	76.63
Hebrew$_{\text{HTB}}$	equal PoS	$120_{\pm 0}$	63.04	58.66	$70_{\pm 1}$	63.56	58.80
Kazakh$_{\text{KTB}}$	XPOS	$283_{\pm 3}$	32.93	17.07	$178_{\pm 5}$	23.09	12.73
Tamil$_{\text{TTB}}$[5]	UPOS	$150_{\pm 2}$	71.59	64.00	$127_{\pm 3}$	75.41	68.58

Table 6: Comparison of the relative PoS-based models against the transition-based BIST parser in terms of accuracy and speed on the UD-CoNLL18 test sets.

In general, our sequence labeling parser outperforms BIST parser in 3 out of 8 treebanks in terms of LAS scores and yields higher parsing speed in all instances. It is also worth noting that our model obtains better accuracy on the highly non-projective treebank in Ancient Greek. This is likely to be strongly related to the fact that the relative PoS-based encoding can fully handle non-projectivity while the variant of the BIST parser used in the experiment is fully projective.

3.3.2 Further advances using Multi-task learning

As mentioned before, each label can be decomposed into atomic elements. Hence, in this experiment we want to examine whether learning such labels using MTL may lead to further improvements in accuracy and parsing speed of our sequence labeling parser.

Model As shown in the previous experiment, a large label set size can impede efficient learning due to the sparsity making the classification harder. Hence, instead of forcing the model to learn the entire labels and possibly introducing sparsity, an alternative is to have the model learn the best combination of the components for each label. This can be obtained by applying MTL, where each component is defined as a separate task t and the model is optimized by computing loss as $\mathcal{L}{=}\sum_t \mathcal{L}_t$, where $\mathcal{L}_t$ is the partial loss of each subtask t. Figure 11 shows architectures corresponding to each task definition.

More specifically, a label obtained with the relative PoS-based encoding can be decomposed into its elements (o_i, p_i, r_i), where each component or a combination of them can be learned as a separate task. The BiLSTM representations are shared across the model and then each task corresponds to a separate feed-forward layer with a *softmax*. This has a practical implication. When learning a label as a single task, we construct a *softmax* corresponding to the label set size which in some cases can be very large and sparse. When decomposing each PoS-based

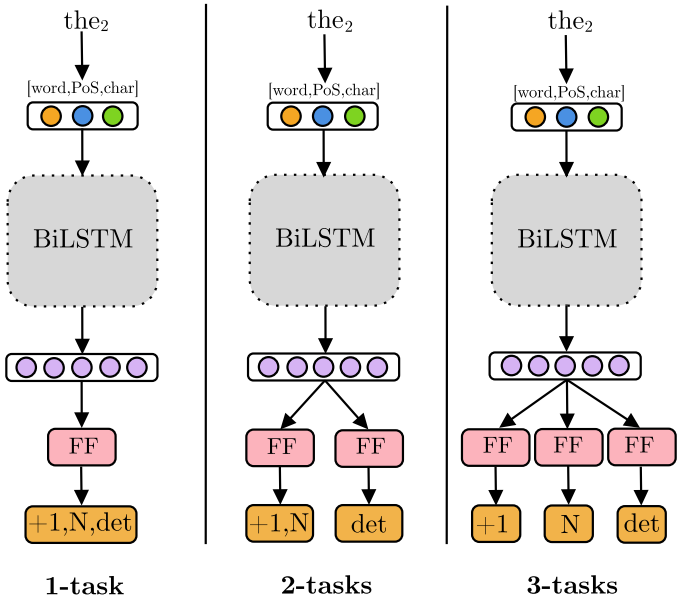


Figure 11: Architectures for dependency label learning using MTL. A label can be defined as a single task or with a label decomposition as multiple subtasks.

label into subtasks, we construct three *softmax* with the corresponding sizes of o, p, r. This, in turn, facilitates the prediction, since not all components combine and instead of producing a label set size of the Cartesian product, it results in $o + p + r$. From the theoretical point of view, models that learn labels as n-tasks, where $n > 1$, should tackle the sparsity problem better (by leaving it up to the model to learn the label combinations) and potentially increase the parsing speed (due to smaller *softmaxes*). In such a setup each task is weighted equally. The hyperparameters are detailed in Section A.1.2.

Setup We train three models that rely on the relative PoS-based encoding, where labels are learned as one, two or three tasks with the architectures shown in Figure 11. Unlike the previous experiment, we add extra labels preceding and following each label sequence for a given sentence, such as BOS (denoting beginning of a sentence) and EOS (end of a sentence), since previous work found it beneficial (Gómez-Rodríguez and Vilares, 2018). The models are tested on the English PTB treebank and the speeds are measured on a single core of the CPU.

Accuracy and parsing speed Table 7 shows UAS and LAS scores obtained by the models with different task assignments. The results indicate that learning PoS-based labels as two tasks yields the best results. This corresponds to labels of the form (o_i, p_i) and (r_i), where the first task captures the relative distance (o_i) together with the head's PoS (p_i), while the second task consists in predicting the dependency

Model	UAS	LAS
1-TASK	93.81	91.59
2-TASKS	**94.03**	**91.78**
3-TASKS	93.66	91.47

Table 7: Comparison of models' accuracy on the PTB dev set. The models are based on the relative PoS-based encoding, where the labels are learned as a single, two and three tasks.

relation type. Intuitively, such a task assignment seems reasonable since o_i and p_i are closely related and jointly form an unlabeled arc. Interestingly, the 3-task model performs worst indicating that learning combinations of all atomic labels separately is more challenging.

Furthermore, Table 8 shows that the improved accuracy of the PoS-based encoding learned as two tasks does not come at any additional computational cost. In fact, it processes a few more sentences per second than the counterpart, plausibly due to the use of smaller *softmaxes*.

Model	(sent/s)	UAS	LAS
1-TASK	$102_{\pm 6}$	93.60	91.74
2-TASKS	$128_{\pm 11}$	93.84	91.83

Table 8: Comparison of parsing speed and accuracy of the 1- and 2-task models based on the relative PoS-based encoding evaluated on the PTB test set.

Later, we will also show that a sequence labeling parser can further leverage the MTL approach by including auxiliary tasks retrieved from complementary data.

3.4 Strengths and limitations

We have presented three encodings that address dependency parsing as a head selection problem in a sequence labeling setup and we have analyzed the peculiarities that each of them exhibits. In general, the encodings differ in the number of labels they generate. Especially, the naive positional encoding seems to be prone to produce a large label space compared to the other encodings. This is due to the fact that it encodes each dependency arc in terms of the head's absolute position which highly varies across sentences. As a result, even the same dependency type (for instance each arc between a determiner and a noun) will be represented by many labels due to the words' absolute positional variance. As a consequence, a higher degree of sparsity may be introduced that equates with a long tail with low frequent labels. This is mitigated in the two other head selection encodings by capturing the head position in a relative way. Although the naive positional encoding

provides a very straightforward approach to encode a dependency tree, it yields significantly less competitive results than its counterparts.

The relative positional encoding, in turn, has been successfully applied in previous work. One of the assets it offers is a compact representation in terms of the smallest number of entire labels. Additionally, unlike the relative PoS-based encoding, it does not rely on any additional features but still performs worse than the latter.

The relative PoS-based encoding achieves the highest score among the head selection encodings showing that this encoding is indeed capable of obtaining competitive results when relying on a neural network architecture in contrast to the previous work (Spoustová and Spousta, 2010). The results also suggest that the model based on this encoding is able to leverage the representations better than the counterpart models due to its good balance between sparsity and learnability. One of the restraints, however, is that this encoding requires PoS tags during parsing. This strong dependence on PoS tags can be perceived as a limitation since their quality and quantity may vary across languages. Especially, the issue may arise in the case of low-resource languages, in which PoS tags may be not available or their prediction accuracy may be low. In practical terms, it requires verifying the PoS tags for a given language beforehand, although relying on UPoS tags from the UD treebanks should provide moderate results for most of the languages. Moreover, this encoding contradicts the recent findings that question the utility of PoS tags in dependency parsing (Lhoneux et al., 2017; Anderson and Gómez-Rodríguez, 2020b). Furthermore, in the context of efficiency, the need of computing PoS tags may additionally slow down the parser. Since this encoding achieves the highest performance, we will use it as a baseline in the next chapters.

3.5 Conclusions

In this chapter, we have presented three encodings belonging to the family of head selection, where the head can be encoded via its absolute position or the relative distance from the focus word. First, we have provided a formulation of each encoding and described the decoding process including postprocessing methods for incorrect labels. Furthermore, we have tested the encodings empirically and compared our models to the related work. Additionally, we have explored various facets of each encoding. The first experiment has shown that the encodings generate varying label set sizes and differ in their label distribution. Moreover, we have investigated the role of PoS tags in the relative PoS-based encoding showing that this type of encoding is sensitive to the quality and quantity of the available PoS tags in a given treebank. In the second experiment we have decomposed each label of the relative PoS-based encoding into its components and applied MTL, such that each component or a combination of them was learned as a

separate task. As a result, the model with the relative PoS-based encoding, in which labels were learned as two tasks, performed best while maintaining the computational efficiency. Finally, we have discussed the assets and limitations of each encoding.

In the next chapter, we will present an alternative family of encodings that represents dependency arcs in terms of balanced bracketing elements.

4 Bracketing-based encodings

The bracketing-based family of encodings relates to the work of Yli-Jyrä (2012) and Yli-Jyrä and Gómez-Rodríguez (2017) and expresses dependency arcs in terms of balanced bracketing elements, such that left arcs are represented with pairs of symbols `<`, `\` and right arcs with `/`, `>`. Unlike the head selection family that encodes the head position directly or in a relative way in the label, bracketing-based encodings generate labels where each of them contains a sequence of bracketing elements representing incoming and outcoming arcs.

We will present two types of bracketing-based encodings, where the first one is a relaxed 1-planar encoding, while the second type is a variant augmented with additional bracketing elements to integrate the property of 2-planarity that enables encoding non-projective arcs.

4.1 Encodings

4.1.1 Relaxed 1-planar bracketing-based encoding

Each label in the relaxed 1-planar bracketing-based encoding is of the form $l_i = (e_i, r_i)$, where e_i holds a sequence of bracketing elements following the regular expression `(<)?((\)*|(/)*)(>)?`, while r_i encodes the dependency relation. More specifically, the bracketing elements denote:

- `<` w_{i-1} has an incoming arc from the right
- `\` w_i has an outgoing arc towards the left. There can be k copies of this character.
- `/` w_{i-1} has an outgoing arc towards the right. There can be k copies of this character.
- `>` w_i has an incoming arc from the left

A left dependency arc from a word w_j to w_i is encoded by a pair of brackets (`<`,`\`) in the label components e_{i+1} and e_j, while a right arc from a word w_i to w_j is captured by a pair of brackets (`/`,`>`) in the label components e_{i+1} and e_j. One may observe that the bracketing elements `<` and `/` are shifted by one position in the label components. For instance, as shown in Figure 12 the information about the token $Kyrie_1$ having an incoming arc from the right (`<`) is encoded in the

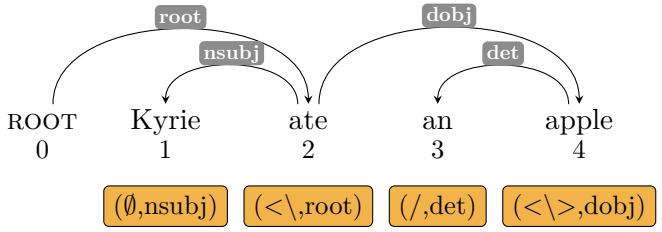


Figure 12: A dependency tree represented with the baseline 1-planar bracketing-based encoding. The ROOT node is not explicitly encoded in any label in order to ease learning.

label corresponding to the next token ate_2. The reason for that is that such a shifting results in a smaller vocabulary size. Without the shift, a label could hold any possible combinations of n outgoing left arcs and n outgoing right arcs making it quadratic in the number of arcs. This is not the case when assuming projectivity and shifting, since each word can either have `\` symbols or `/` symbols, but not both (i.e. `\/`), reducing the number of possible labels from quadratic to linear. However, it is also worth noting that the encoding was initially designed for supporting only projective trees but later on we discovered that it is indeed able to encode some non-projective arcs. That is why we call this 1-planar encoding *relaxed*, since it covers some crossing arcs.

The encoding can be explained more intuitively based on Figure 12. For instance, the token $apple_4$ is assigned the label `<\>` that can be interpreted as: the previous word an_3 has an incoming arc from the right (`<`) that matches the outgoing left arc of $apple_4$ (`\`). Thereby, they form a left dependency arc (`<\`). The remaining incoming arc from the left (`>`) will be matched with the closest outgoing right arc from one of the previous words (in this case ate_2 is the head). Moreover, the arc originating from the ROOT node is not explicitly encoded in order to ease the learning. Later, we will show how the decoding of the labels is performed using separate stacks corresponding to each arc direction.

As mentioned earlier, this encoding was initially designed to only support projective trees, however we detected that it is, in fact, able to cover some non-projective arcs. More concretely, it can preserve crossing arcs in the opposite directions when using separate stacks that balance bracketing elements in the same direction independently. Figure 13 shows an example of a non-projective tree that can be successfully encoded and decoded without any loss of information. Contrarily, in Figure 14, non-projective arcs in the same direction are decoded as projective ones. This limitation stems from the fact that each opening bracket is matched with the closest closing bracket precluding matching the non-projective arcs in the same direction correctly. To alleviate this restricted coverage of non-projectivity, we introduce a variant that makes use of the property of 2-planarity.

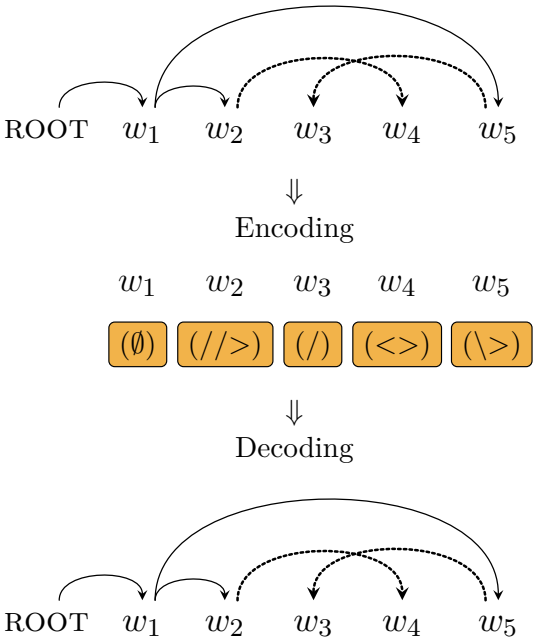


Figure 13: Encoding and decoding of a non-projective tree with the relaxed 1-planar bracketing-based encoding. Since non-projective arcs (denoted with dotted lines) appear in the opposite directions, non-projectivity can be fully preserved during decoding.

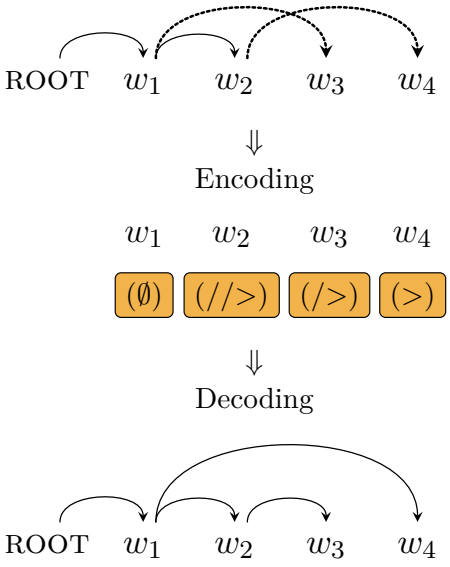


Figure 14: Encoding and decoding of a non-projective tree with crossing arcs in the same directions. Since the relaxed 1-planar bracketing encoding is not able to preserve that information, non-projective arcs are decoded as projective ones.

4.1.2 2-Planar bracketing-based encoding

We will tackle the restricted coverage of non-projectivity by proposing a variant that makes use of the notion of 2-planarity that has been earlier explored in the context of dependency parsing (Yli-Jyrä, 2003; Gómez-Rodríguez and Nivre, 2010; Gómez-Rodríguez and Nivre, 2013). First, we will introduce the concept of k-planarity followed by the encoding definition. Furthermore, we will describe two plane assignment strategies that can be applied.

4.1.2.1 k-Planarity

A non-projective tree can be often decomposed into a set of subgraphs, such that each contains only projective dependencies. This can be obtained by applying the property of *k-planarity*, where edges of a non-projective tree are partitioned into k planes (sets) such that arcs in the same plane do not cross. More formally, following Gómez-Rodríguez and Nivre (2013) k-planarity can be defined as:

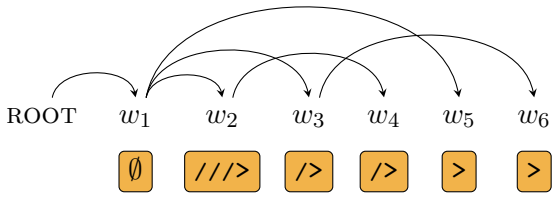
- K-PLANARITY if $\exists(G_1 = (V, A_1), \ldots, G_k = (V, A_k))$ for $k \geqslant 1$: $A = A_1 \cup \ldots \cup A_k$ where arcs $\in A_k$ do not cross

However, partitioning arcs into k planes can be often done in a number of ways. Therefore, it is desirable to find a systematic method that assures selecting the most optimal partitioning of a dependency tree. Hence, we propose two alternative plane assignment strategies for 2-planar dependency trees. The reason for operating on 2 planes is that, as Gómez-Rodríguez and Nivre (2010) demonstrate, the vast majority of dependency structures can be projected in a 2-planar manifold. Moreover, two planes suffice to obtain a good accuracy and efficiency and using more planes increases the complexity of a parser without providing considerable accuracy gains (Gómez-Rodríguez and Nivre, 2013). Now, we will show how the notion of 2-planarity can be embedded in sequence labeling parsing.

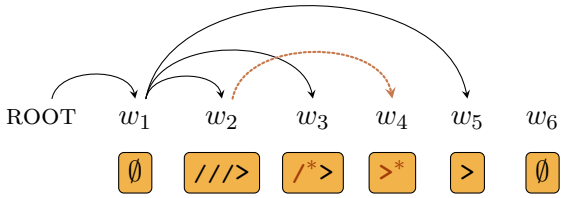
4.1.2.2 2-Planar labeling

In the relaxed 1-planar bracketing-based encoding, a label for each word represents a sequence of bracketing elements belonging to a set $B = \{<, \backslash, /, >\}$ projected on a single plane. In order to augment this approach with the property of 2-planarity, we introduce a set of "star" bracketing elements from $B^* = \{<^*, \backslash^*, /^*, >^*\}$ denoting membership to the second plane. Each label can hold elements from both B and B^*, however during decoding they are placed on separate stacks. Figure 15 illustrates a non-projective tree encoded with the relaxed 1-planar bracketing-based encoding and with the 2-planar variants under

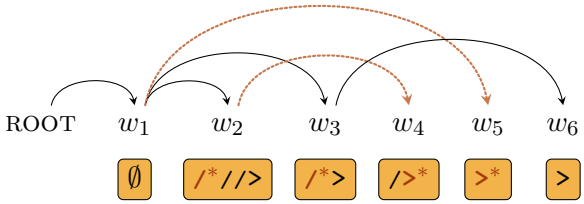
two different plane assignment strategies, which will be explained below. Note that the 1-planar encoding shown in (a) would result in a truncated tree without crossing arcs after decoding, like in the previous example in Figure 14, while the 2-planar encodings shown in (b) and (c) allow us to recover the tree as shown in the picture.



(a) Bracketing-based encoding of a non-projective tree cast on a single plane.



(b) 2-planar bracketing-based encoding with second-plane-averse greedy plane assignment.



(c) 2-planar bracketing-based encoding with plane assignment based on restriction propagation on the crossings graph.

Figure 15: A non-projective tree represented with 1- and 2-planar bracketing-based encodings. The dotted arcs pertain to the second plane denoted with * in the label.

As mentioned earlier, there are often multiple ways of assigning arcs to each plane resulting in various partition schemes. Since the presence of crossing arcs in dependency treebanks is rather scarce (Ferreri-Cancho, Gómez-Rodríguez, and Esteban, 2018), we define plane assignment strategies that will minimize the use of the second plane only when it is strictly necessary. Thereby, we will diminish the amount of additional labels with respect to the relaxed 1-planar bracketing-based encoding in order to avoid sparsity that could impede an efficient label learning. On these grounds, we introduce two plane assignment strategies in which arcs are traversed from left to right of their right endpoint, with the shortest arc first for arcs with a common right endpoint.

4.1.2.3 Second-Plane-Averse greedy plane assignment strategy

In this strategy, we attempt to assign each arc a primarily to the first plane P_1 provided that no other arc crossing a has already been assigned to that plane. Otherwise, we project the arc a on the second plane P_2, unless there are arcs crossing a that already lie on the first and second plane. In the latter case, we do not assign any plane to the arc a. More formally, this strategy can be described with pseudocode shown in algorithm 1 and an example of this encoding is illustrated in (b) in Figure 15. It is worth noting that according to this strategy the arc $w_2 \to w_4$ in the figure is projected on the second plane, while no plane is assigned to the arc $w_3 \to w_6$, since it would cross arcs pertaining to both first and second plane. Even though this tree is indeed 2-planar, the greedy plane assignment strategy is not able to cover all arcs that in theory can be projected into two planes. Hence, we will now introduce a second plane assignment strategy that ensures a complete encoding of 2-planar trees.

Algorithm 1: 2p-greedy

```
Input: A set of arcs T, and input length n
Result: Two sets (planes) of arcs P_1, P_2
P_1 ← ∅;
P_2 ← ∅;
for x_r ← 1 to n do
    for x_l ← x_r − 1 downto 0 do
        if ∃a ∈ T | a = (x_l, x_r, l) ∨ a = (x_r, x_l, l) then
            nextArc ← a;
            C ← {b ∈ (P_1 ∪ P_2) | b crosses a};
            if C ∩ P_1 = ∅ then
                P_1 ← P_1 ∪ {nextArc};
            else if C ∩ P_2 = ∅ then
                P_2 ← P_2 ∪ {nextArc};
            else
                do nothing (failed to assign nextArc to a plane);
        end
    end
end
return P_1, P_2;
```

4.1.2.4 Plane assignment strategy based on restriction propagation on the crossings graph

In the previous strategy, the plane assignment process is performed greedily, where the scope for forbidding arcs from a plane is restricted

Algorithm 2: 2p-prop

Input: A set of arcs T, and input length n
Result: Two sets (planes) of arcs P_1, P_2
function `Propagate(`*Edge sets* $T, \overline{P_1}, \overline{P_2}$*, Edge* e*, Plane* i`):`
  $\overline{P_i} \leftarrow \overline{P_i} \cup \{e\}$;
  // e forbidden from plane i
  for *(*$e' \in T \mid e'$ crosses e*)* **do**
    if $e' \notin \overline{P_{3-i}}$ **then**
      $(\overline{P_1}, \overline{P_2}) \leftarrow$ `Propagate(`$T, \overline{P_1}, \overline{P_2}, e', 3-i$`)`;
  end
  return $\overline{P_1}, \overline{P_2}$;
$P_1 \leftarrow \varnothing$, $P_2 \leftarrow \varnothing$, $\overline{P_1} \leftarrow \varnothing$, $\overline{P_2} \leftarrow \varnothing$;
for $x_r \leftarrow 1$ ***to*** n **do**
  for $x_l \leftarrow x_r - 1$ ***downto*** 0 **do**
    if $\exists a \in T \mid a = (x_l, x_r, l) \vee a = (x_r, x_l, l)$ **then**
      nextArc $\leftarrow a$;
      if nextArc $\notin \overline{P_1}$ **then**
        $P_1 \leftarrow P_1 \cup \{\text{nextArc}\}$;
        `Propagate(`$T, \overline{P_1}, \overline{P_2}$,*nextArc,*2`)`;
      else if nextArc $\notin \overline{P_2}$ **then**
        $P_2 \leftarrow P_2 \cup \{\text{nextArc}\}$;
        `Propagate(`$T, \overline{P_1}, \overline{P_2}$,*nextArc,*1`)`;
      else
        do nothing (failed to assign nextArc to a plane);
    end
  end
end
return P_1, P_2;

to the locally involved crossing arcs with respect to a given arc. This can lead to suboptimal decisions resulting in leaving out some arcs in graphs that, in fact, have 2-planar representations.

To assure a full coverage of 2-planar structures, we introduce a plane assignment strategy based on restriction propagation on the crossings graph, in which two nodes are linked if their corresponding edges cross (Gómez-Rodríguez and Nivre, 2013). The idea behind this strategy is that one may anticipate that the locally involved crossing arcs may also collide with their neighboring arcs. Thus, the plane restriction can be propagated to all subsequent crossing arcs originating from a given arc. Similarly as in the previous strategy, the process proceeds by trying to assign an arc to the first plane P_1 if allowed, otherwise to the second P_2 and if it is not possible, no plane is assigned to that arc. However, the difference lies in propagating restriction on all involved arcs. Specifically, assigning an arc to the first plane implies that we forbid all its crossing arcs from that plane $\overline{P_1}$, then the neighbors of

that arcs are forbidden from the second plane $\overline{P_2}$ and so forth. The strategy is described with a pseudocode in algorithm 2. As shown in (c) in Figure 15, this method, unlike the previous strategy, is able to encode the arc $w_3 \rightarrow w_6$ providing a full coverage of arcs in the tree.

4.1.2.5 Other plane assignment strategies

There are alternative plane assignment strategies which can be applied instead of averting from using the second plane as in this work. For instance, one can define a strategy that minimizes switching between the planes. In other words, instead of continuously attempting to switch to the first plane, the aim is to hold to the same plane. In this vein, Gómez-Rodríguez and Nivre (2010) apply a switch-averse restriction-propagation strategy in a 2-planar transition-based parser. This strategy is especially useful since it minimizes the number of transitions. However, this advantage vanishes in the case of dependency parsing cast as sequence labeling. We have included the switch-averse strategy in our initial experiments, however it led to slightly worse performance and as a result, we have discarded it. In general, one may apply any other strategy based on some other preference criteria. In our approach, we choose the two previously described strategies since they facilitate keeping the vocabulary size small.

4.2 Decoding

Now we will describe the decoding process for the relaxed 1-planar bracketing-based encoding and the 2-planar variants.

4.2.1 Relaxed 1-planar bracketing-based decoding

The decoding process of the relaxed 1-planar bracketing-based encoding consists in finding pairs of matching bracketing elements, where the pairs are of the form (<,\) and (/,>) associated with a left and a right arc, respectively. Initially, two empty stacks are created: σ_L and σ_R that hold bracketing elements separately according to the arc direction. We process the output labels in a left-to-right fashion and decompose them into bracketing elements, such that any < and \ are processed in σ_L, while / and > in σ_R.

The decoding proceeds by reading the label for each token. If the element in the label of a token w_i is a left opening bracket (<), we push it to σ_L preserving the index i. Once having read a closing bracket (\) for the word w_j, we associate it with an opening bracket on the top of the stack σ_L, which is then popped. In other words, if the most recently read opening bracket corresponds to the index i, a left-arc is created from w_j to w_{i-1}. For right arcs, an analogous processing takes

place but in the stack σ_R. In the case of predicted bracketing elements not being well-balanced, the outermost elements are discarded. Finally, we apply the postprocessing heuristics to recover arcs that could not be successfully decoded due to the incorrect labels (see Section 3.2.1).

4.2.2 2-Planar bracketing-based decoding

Decoding of the 2-planar bracketing-based labels proceeds similarly as in the 1-planar encoding, where we search for pairs of matching bracketing elements (<,\) and (/,>) belonging to the first plane. However, additionally we look for pairs (<*,*) and (/*,>*) pertaining to the second plane. Initially, four empty stacks are created, where σ_L and σ_R hold left and right arcs from the first plane, while σ_L^* and σ_R^* stacks handle bracketing elements from the second plane. The decoding process continues analogously to the one described above, where the outermost elements from the unbalanced components in the first or second plane are removed, followed by the postprocessing steps.

4.3 Analysis

In this section, we will provide an in-depth analysis of the bracketing-based family of encodings, while the final evaluations of the models will be provided in the subsequent section. We will first examine the facets of the relaxed 1-planar bracketing-based encoding, referred to as 1P-BRACKETS, followed by an analysis of its 2-planar variants, denoted as 2P-GREEDY and 2P-PROP, respectively. Since the relative PoS-based encoding achieved the best performance among the head selection encodings, we select it as our baseline encoding that will serve for comparison in the following experiments.

4.3.1 Aspects of the relaxed 1-planar bracketing-based encoding

The relaxed 1-planar bracketing encoding will be assessed by taking into account several aspects. First, we will compare the encodings with respect to the models‘ efficiency. Next, we will examine their label set sizes and distributions.

4.3.1.1 Model efficiency

In this experiment, we want to explore which sequence labeling model preserves the most optimal trade-off between the speed and accuracy. To do so, we will compute the Pareto frontier for the relative PoS-based and the relaxed 1-planar bracketing-based models with different hyperparameter combinations.

Model As previously, the experiment is carried out by relying on NCRF++ with BiLSTMs, where an entire label is learned as a single task. The parameter search is performed by altering the number of hidden layers, their dimensions and presence of character embeddings, as they influence the speed and accuracy of a model. More specifically, we test the following models specified in Table 9 with remaining hyperparameters listed in Section A.1.1.

Model	1	2	3	4	5	6	7	8	9	10
#BiLSTM layers	1	2	2	2	2	2	2	2	3	3
Hidden vector dim.	800	250	250	400	600	800	1000	1200	400	800
Char embeddings	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓

Table 9: Hyperparameter search for model selection.

Setup We train ten models relying on the relative PoS-based encoding and another ten models with the relaxed 1-planar bracketing-based encoding. We will refer to the the relative PoS-based models as $P^{z}_{x,y}$ while to the relaxed 1-planar bracketing-based models as $B^{z}_{x,y}$, where z denotes whether character embeddings are used, x the number of BiLSTM layers, while y stands for the word hidden vector dimensions. We evaluate all models on PTB with its common setup and we measure the parsing speed in sentences per second on a single core of CPU (Intel Core i7-7700 CPU 4.2 GHz) and on a GPU (GeForce GTX 1080).

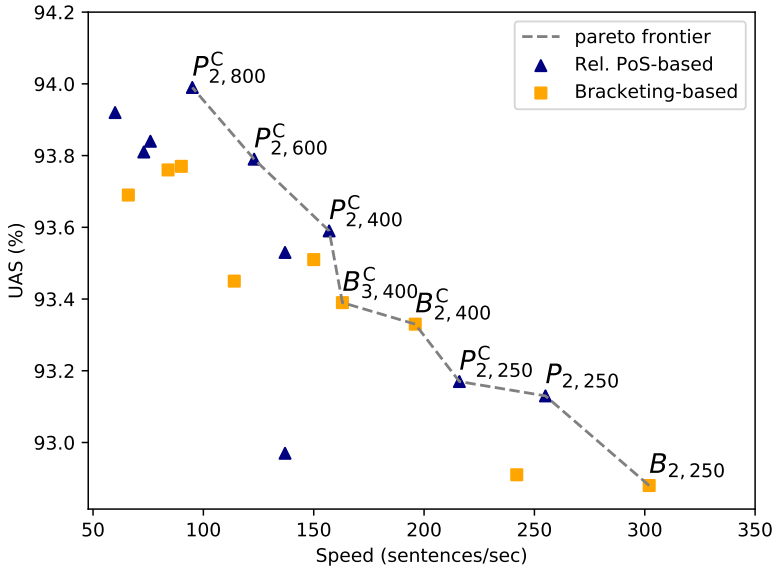


Figure 16: UAS/speed Pareto frontier for models with the relative PoS-based and the relaxed 1-planar bracketing-based encodings based on various hyperparameter combinations. The evaluation is performed on the PTB dev set.

Pareto frontier In order to determine the efficiency of the models in terms of speed and accuracy, we calculate the Pareto frontier. As shown in Figure 16, we obtain eight models that lie in the frontier preserving the best trade-off between speed (sentences/second processed on a single core CPU) and UAS score. Among these, three are based on the relaxed 1-planar bracketing encoding that are located in the center or below the Pareto frontier. In general, the most efficient models rely on either 2 or 3 BiLSTM layers and with hidden vector dimensions in the range 250-800, while the vast majority also uses character embeddings.

4.3.1.2 Label distribution and label set size

Label distribution To gain a deeper understanding of the differences between the relaxed 1-planar bracketing-based and the relative PoS-based encodings, we examine their distribution of labels on the English PTB dev set. More specifically, we look at the distribution of the first component of each label e_i containing the head-related information (the relative distance based on the head's PoS or a sequence of bracketing elements excluding the components for dependency relation that have the same size in all encodings). As illustrated in Figure 17, the bracketing based-encoding tends to generate fewer e_i labels than its counterpart. It is also discernible that the relative PoS-based encoding has a longer tail with low frequency labels indicating that the bracketing-based encoding provides a more compact label vocabulary.

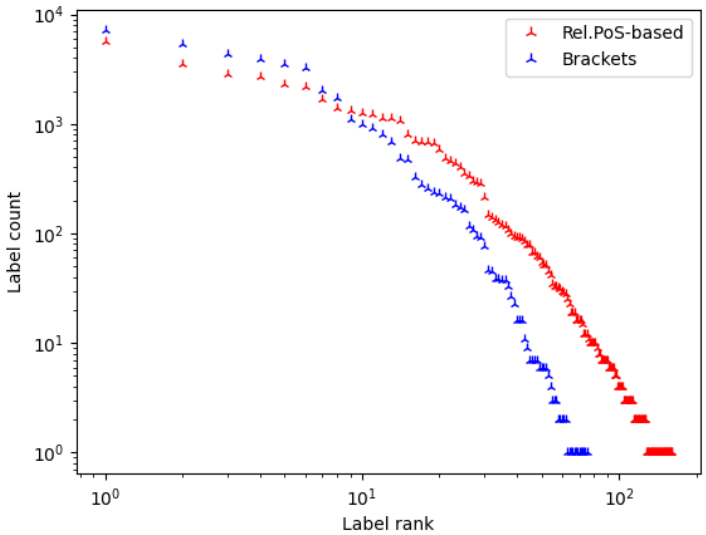


Figure 17: Distribution of label frequency rank of e_i with respect to the occurrences in the PTB dev set represented in log-log scale.

Treebank	Encoding	Task 1 #labels e_i	Task 2 #labels r_i
Ancient Greek$_{\text{Perseus}}$	rel-PoS	166	27
	1p-brackets	210	27
Chinese$_{\text{GSD}}$	rel-PoS	201	45
	1p-brackets	104	45
English$_{\text{EWT}}$	rel-PoS	202	51
	1p-brackets	109	51
Finnish$_{\text{TDT}}$	rel-PoS	220	47
	1p-brackets	107	47
Hebrew$_{\text{HTB}}$	rel-PoS	162	40
	1p-brackets	107	40
Russian$_{\text{GSD}}$	rel-PoS	169	44
	1p-brackets	93	44
Tamil$_{\text{TTB}}$	rel-PoS	68	31
	1p-brackets	49	31
Uyghur$_{\text{UDT}}$	rel-PoS	102	44
	1p-brackets	58	44
Wolof$_{\text{WTB}}$	rel-PoS	102	40
	1p-brackets	69	40

Table 10: Comparison of label set sizes generated by the relative PoS-based and bracketing-based encodings considered as two tasks. e_i refers to the head-related label, while r_i to the label with the dependency relation type.

Label set size To extend the analysis to a wider range of languages, we look at the label set size that both encodings generate on a representative subset of UDv2.4 (Nivre et al., 2019) treebanks. Table 10 compares the label set sizes of both encoding types. In general, a similar pattern emerges, where the bracketing-based encoding is more compact. More specifically, the relative PoS-based encoding produces on average roughly 1.5x more labels than its counterpart in the selected treebanks. As mentioned earlier, the dependency relation labels are identical for both encodings, hence the labels for the second task are of the same size.

4.3.2 Aspects of the 2-planar bracketing-based encodings

Unlike the relative PoS-based encoding, the relaxed 1-planar bracketing-based encoding is not able to fully encode non-projective arcs. Hence, to mitigate the limitation of this encoding, we enhance it with the notion of 2-planarity. In this section, we will examine the improvements in the coverage of non-projective arcs obtained by the 2-planar encodings evaluated on a set of highly non-projective treebanks. Additionally, we will compare their label set size and label coverage at testing time.

Model The encodings are learned with a BiLSTM-based model. More specifically, the 2-planar information is incorporated using MTL, where a model has three separate output layers: one for predicting labels for the first plane, another for the second plane and one for the dependency relation types. All tasks are equally weighted. To ensure a fair comparison against the relaxed 1-planar bracketing-based encoding, we learn the latter as two tasks corresponding to the labels for the first plane and dependency relation types. The hyperparameters are detailed in Section A.1.1.

Setup To test the ability of handling non-projectivity of the 2-planar encodings, we extract a set of UDv2.4 treebanks (Nivre et al., 2019) that contain the highest percentage of non-projective sentences. For the experiment, we discard some treebanks due to the lack of a development set or a pre-trained UDPipe model, resulting in the following treebanks: Ancient Greek$_{\text{Perseus}}$, Basque$_{\text{BDT}}$, Hungarian$_{\text{Szeged}}$, Portuguese$_{\text{Bosque}}$, Urdu$_{\text{UDTB}}$, Afrikaans$_{\text{AfriBooms}}$, Korean$_{\text{Kaist}}$, Danish$_{\text{DDT}}$, Gothic$_{\text{PROIEL}}$ and Lithuanian$_{\text{HSE}}$. In addition, we include two control treebanks that are fully projective: Galician$_{\text{CTG}}$ and Japanese$_{\text{GSD}}$. Table 11 lists the selected treebanks with the corresponding percentages of non-projective sentences and dependencies.

Language	% non-projective sentences	% non-projective dependencies
Ancient Greek$_{\text{Perseus}}$	63.87	10.14
Basque$_{\text{BDT}}$	33.17	4.69
Hungarian$_{\text{Szeged}}$	27.11	1.97
Portuguese$_{\text{Bosque}}$	23.31	1.85
Urdu$_{\text{UDTB}}$	22.57	1.32
Afrikaans$_{\text{AfriBooms}}$	22.34	1.62
Korean$_{\text{Kaist}}$	21.70	2.55
Danish$_{\text{DDT}}$	21.50	1.74
Gothic$_{\text{PROIEL}}$	17.57	2.53
Lithuanian$_{\text{HSE}}$	17.49	1.27
Japanese$_{GSD}$	*0*	*0*
Galician$_{CTG}$	*0*	*0*

Table 11: Comparison of non-projective sentences and dependencies (%) across the selected UD treebanks. Japanese$_{\text{GSD}}$ and Galician$_{\text{CTG}}$ are fully projective and serve as control treebanks.

Among the selected treebanks, the most non-projective one is Ancient Greek containing almost 64% non-projective sentences and around 10% non-projective dependencies. However, it is discernible that in the remaining treebanks the degree of non-projectivity is considerably lower

that may translate into smaller accuracy gains. More detailed statistics about the selected treebanks, such as the percentage of non-projective sentences in each data split, are provided in Section B.1.

4.3.2.1 Non-projectivity coverage

Now, we move on to the theoretical upper bound of the 2-planar encodings by reconstructing the gold training sets. Table 12 shows the arc coverage percentage obtained by each encoding. In general, both 2-planar encodings provide more than 99% arc coverage from the gold data sets and as expected, they preserve more non-projective arcs than the relaxed 1-planar bracketing-based encoding. However, the latter is still able to moderately reconstruct the data sets, even in such a highly non-projective treebank as Ancient $\text{Greek}_{\text{Perseus}}$ (89.53% of coverage against over 99.2% obtained by the 2-planar variants). It may suggest that the treebank contains crossing arcs that to a large extent point in the opposite direction and hence can be handled by this encoding.

When comparing solely the 2-planar encodings based on different plane assignment strategies, their arc coverage is on par. One could expect 2P-PROP to achieve better performance than 2P-GREEDY due to its full coverage of 2-planar trees. However, in some languages there may be a fraction of trees that are not 2-planar, hence the property of 2P-PROP does not apply and as a result, 2P-GREEDY may cover in some cases more trees.

Language	1p-brackets	2p-greedy	2p-prop
$\text{Ancient Greek}_{\text{Perseus}}$	89.53	99.27	**99.33**
$\text{Basque}_{\text{BDT}}$	94.85	**99.85**	99.62
$\text{Hungarian}_{\text{Szeged}}$	97.57	99.96	**99.98**
$\text{Portuguese}_{\text{Bosque}}$	98.10	**99.95**	99.88
$\text{Urdu}_{\text{UDTB}}$	98.68	**99.95**	99.94
$\text{Afrikaans}_{\text{AfriBooms}}$	98.65	**99.99**	**99.99**
$\text{Korean}_{\text{Kaist}}$	98.42	**100.00**	**100.00**
$\text{Danish}_{\text{DDT}}$	98.10	**99.97**	99.96
$\text{Gothic}_{\text{PROIEL}}$	97.58	99.94	**99.98**
$\text{Lithuanian}_{\text{HSE}}$	98.35	99.97	**100.00**

Table 12: Upper bound for arc coverage percentage of the 1- and 2-planar encodings on the gold training set across highly non-projective UD treebanks.

Next, we analyze the precision and recall of sequence labeling parsing models trained to use the 1- and 2-planar bracketing-based encodings. Table 13 reports the precision and recall on *non-projective sentences*, while Table 14 on *non-projective dependencies*. The accuracy

Language	1p-brackets		2p-greedy		2p-prop	
	P	R	P	R	P	R
Ancient Greek$_{\text{Perseus}}$	85.74	54.34	86.33	63.85	**87.58**	**66.23**
Basque$_{\text{BDT}}$	69.87	45.80	70.14	**52.97**	**72.77**	52.80
Hungarian$_{\text{Szeged}}$	37.17	66.98	35.51	71.70	**37.80**	**74.53**
Portuguese$_{\text{Bosque}}$	52.94	24.77	55.84	39.45	**61.64**	**41.28**
Urdu$_{\text{UDTB}}$	36.63	**36.63**	38.10	31.68	**39.78**	**36.63**
Afrikaans$_{\text{AfriBooms}}$	40.99	65.35	46.72	63.37	**46.94**	**68.32**
Korean$_{\text{Kaist}}$	59.45	**49.54**	62.24	47.03	**62.80**	47.03
Danish$_{\text{DDT}}$	45.54	**48.57**	**46.36**	**48.57**	45.37	46.67
Gothic$_{\text{PROIEL}}$	50.50	26.42	**58.88**	32.64	56.00	**36.27**
Lithuanian$_{\text{HSE}}$	**34.38**	**91.67**	27.59	66.67	33.33	83.33
Average	51.32	51.01	52.77	51.79	**54.40**	**55.31**

Table 13: Precision and recall of the 1- and 2-planar bracketing-based models on non-projective sentences in the test set.

on non-projective sentences is computed by verifying whether a gold non-projective sentence is identified by a model as non-projective, disregarding the correctness of its predicted non-projective dependencies.

Language	1p-brackets		2p-greedy		2p-prop	
	P	R	P	R	P	R
Ancient Greek$_{\text{Perseus}}$	20.82	10.32	**32.40**	18.65	31.40	**19.16**
Basque$_{\text{BDT}}$	18.41	11.80	28.11	19.83	**31.76**	**20.40**
Hungarian$_{\text{Szeged}}$	1.57	4.05	3.13	9.25	**4.06**	**10.98**
Portuguese$_{\text{Bosque}}$	10.87	5.18	14.50	9.84	**20.18**	**11.92**
Urdu$_{\text{UDTB}}$	3.26	**3.92**	0.69	0.65	**3.41**	**3.92**
Afrikaans$_{\text{AfriBooms}}$	11.04	18.09	**13.09**	**19.15**	12.59	18.62
Korean$_{\text{Kaist}}$	28.12	**21.68**	**32.26**	21.37	31.06	21.53
Danish$_{\text{DDT}}$	7.66	11.35	**12.96**	**19.86**	9.45	13.48
Gothic$_{\text{PROIEL}}$	11.11	5.17	**19.63**	11.03	17.46	**11.38**
Lithuanian$_{\text{HSE}}$	0	0	0	0	**1.64**	**6.25**
Average	11.29	9.16	15.68	12.96	**16.30**	**13.76**

Table 14: Precision and recall of the 1- and 2-planar bracketing-based models on non-projective dependencies in the test set.

The results show that both 2-planar encodings perform better than the 1-planar counterpart when predicting non-projective sentences. On average, 2P-PROP yields the highest precision and recall of all encodings

suggesting that it conveys representations that facilitate the identification of non-projective sentences. Regarding the precision and recall on the non-projective dependencies, the 2-planar encodings also perform on average better than the encoding with arcs projected on a single plane. Overall, 2P-PROP achieves slightly higher averaged precision and recall than 2P-GREEDY. Interestingly, $\text{Lithuanian}_{\text{HSE}}$ obtains very low precision and recall on non-projective dependencies compared to the other treebanks. However, the treebank statistics from Section B.1 show that $\text{Lithuanian}_{\text{HSE}}$ contains a small number of sentences, which impedes drawing robust conclusions about its general poor performance.

4.3.2.2 Label set size and label coverage

Label set size Furthermore, we investigate the number of atomic labels per plane that the 2-planar encodings generate on the training and development data sets compared to the other encodings. We remind that since 2P-GREEDY and 2P-PROP models require an additional plane, this implies introducing an extra task that contains the "star" brackets as its output vocabulary. Each task is also augmented with the special BOS and EOS labels to mark the beginning and the end of a sentence. The label set sizes are shown in Table 15.

Language	Encoding	Task 1 (1rst plane)	Task 2 (2nd plane)	Task 3 (deprel)	Language	Encoding	Task 1 (1rst plane)	Task 2 (2nd plane)	Task 3 (deprel)
Ancient $\text{Greek}_{\text{Perseus}}$	rel-PoS	166	–	27	$\text{Korean}_{\text{Kaist}}$	rel-PoS	134	–	32
	1p-brackets	210	–	27		1p-brackets	89	–	32
	2p-greedy	108	37	27		2p-greedy	73	14	32
	2p-prop	109	39	27		2p-prop	73	14	32
$\text{Basque}_{\text{BDT}}$	rel-PoS	132	–	32	$\text{Danish}_{\text{DDT}}$	rel-PoS	150	–	38
	1p-brackets	134	–	32		1p-brackets	128	–	38
	2p-greedy	84	25	32		2p-greedy	97	23	38
	2p-prop	83	25	32		2p-prop	96	25	38
$\text{Hungarian}_{\text{Szeged}}$	rel-PoS	128	–	56	$\text{Gothic}_{\text{PROIEL}}$	rel-PoS	121	–	34
	1p-brackets	101	–	56		1p-brackets	114	–	34
	2p-greedy	71	19	56		2p-greedy	78	18	34
	2p-prop	71	21	56		2p-prop	78	19	34
$\text{Portuguese}_{\text{Bosque}}$	rel-PoS	192	–	43	$\text{Lithuanian}_{\text{HSE}}$	rel-PoS	89	–	38
	1p-brackets	110	–	43		1p-brackets	57	–	38
	2p-greedy	88	25	43		2p-greedy	46	11	38
	2p-prop	88	27	43		2p-prop	46	12	38
$\text{Urdu}_{\text{UDTB}}$	rel-PoS	190	–	27	*Japanese$_{GSD}$*	*rel-PoS*	*77*	–	*27*
	1p-brackets	95	–	27		*1p-brackets*	*45*	–	*27*
	2p-greedy	80	22	27		*2p-greedy*	*45*	*3*	*27*
	2p-prop	80	22	27		*2p-prop*	*45*	*3*	*27*
$\text{Afrikaans}_{\text{AfriBooms}}$	rel-PoS	110	–	28	*Galician$_{CTG}$*	*rel-PoS*	*132*	–	*26*
	1p-brackets	77	–	28		*1p-brackets*	*82*	–	*26*
	2p-greedy	62	15	28		*2p-greedy*	*82*	*3*	*26*
	2p-prop	62	15	28		*2p-prop*	*82*	*3*	*26*

Table 15: Atomic label set size per plane of the relative PoS-based, 1- and 2-planar bracketing-based encodings retrieved from the training and dev set. Each task contains three additional labels: BOS, EOS and $\emptyset$ (empty label).

In general, the bracketing-based encodings require fewer labels in the majority of the treebanks than the relative PoS-based encoding, which is in line with the previous analysis. When comparing the 1- and 2-planar bracketing-based encodings, the latter does not result in

Language	Encoding	Task 1 (1st plane) Unseen	Total	% occ.	Task 2 (2nd plane) Unseen	Total	% occ.	Task 3 (deprel) Unseen	Total	% occ.
Ancient Greek$_{\text{Perseus}}$	rel-PoS	0 (*0%*)	75	0	–	–	–	0 (*0%*)	25	0
	1p-brackets	4 (*3.39%*)	118	0.02	–	–	–	0 (*0%*)	25	0
	2p-greedy	2 (*3.08%*)	65	0.01	1 (*5.26%*)	19	0.01	0 (*0%*)	25	0
	2p-prop	1 (*1.49%*)	67	0	0 (*0%*)	22	0	0 (*0%*)	25	0
Basque$_{\text{BDT}}$	rel-PoS	4 (*4.3%*)	93	0.02	–	–	–	0 (*0%*)	30	0
	1p-brackets	2 (*2.15%*)	93	0.01	–	–	–	0 (*0%*)	30	0
	2p-greedy	3 (*4.35%*)	69	0.02	2 (*10.53%*)	19	0.01	0 (*0%*)	30	0
	2p-prop	3 (*4.35%*)	69	0.02	2 (*10.0%*)	20	0.01	0 (*0%*)	30	0
Hunagrian$_{\text{Szeged}}$	rel-PoS	6 (*7.5%*)	80	0.06	–	–	–	0 (*0%*)	47	0
	1p-brackets	5 (*6.1%*)	82	0.05	–	–	–	0 (*0%*)	47	0
	2p-greedy	1 (*1.64%*)	61	0.01	1 (*8.33%*)	12	0.01	0 (*0%*)	47	0
	2p-prop	1 (*1.64%*)	61	0.01	1 (*6.67%*)	15	0.01	0 (*0%*)	47	0
Portuguese$_{\text{Bosque}}$	rel-PoS	2 (*2.11%*)	95	0.03	–	–	–	0 (*0%*)	38	0
	1p-brackets	0 (*0%*)	60	0	–	–	–	0 (*0%*)	38	0
	2p-greedy	0 (*0%*)	54	0	0 (*0%*)	14	0	0 (*0%*)	38	0
	2p-prop	0 (*0%*)	54	0	0 (*0%*)	14	0	0 (*0%*)	38	0
Urdu$_{\text{UDTB}}$	rel-PoS	8 (*7.69%*)	104	0.07	–	–	–	0 (*0%*)	24	0
	1p-brackets	4 (*6.06%*)	66	0.03	–	–	–	0 (*0%*)	24	0
	2p-greedy	3 (*5.36%*)	56	0.02	0 (*0%*)	12	0	0 (*0%*)	24	0
	2p-prop	3 (*5.36%*)	56	0.02	0 (*0%*)	14	0	0 (*0%*)	24	0
Afrikaans$_{\text{AfriBooms}}$	rel-PoS	2 (*2.82%*)	71	0.02	–	–	–	0 (*0%*)	26	0
	1p-brackets	4 (*6.06%*)	66	0.04	–	–	–	0 (*0%*)	26	0
	2p-greedy	1 (*1.92%*)	52	0.01	1 (*10.0%*)	10	0.01	0 (*0%*)	26	0
	2p-prop	1 (*1.89%*)	53	0.01	1 (*9.09%*)	11	0.01	0 (*0%*)	26	0
Korean$_{\text{Kaist}}$	rel-PoS	1 (*1.11%*)	90	0	–	–	–	1 (*3.23%*)	31	0
	1p-brackets	1 (*1.59%*)	63	0	–	–	–	1 (*3.23%*)	31	0
	2p-greedy	0 (*0%*)	56	0	0 (*0%*)	8	0	1 (*3.23%*)	31	0
	2p-prop	0 (*0%*)	56	0	0 (*0%*)	8	0	1 (*3.23%*)	31	0
Danish$_{\text{DDT}}$	rel-PoS	2 (*2.25%*)	89	0.02	–	–	–	0 (*0%*)	34	0
	1p-brackets	0 (*0%*)	72	0	–	–	–	0 (*0%*)	34	0
	2p-greedy	0 (*0%*)	63	0	0 (*0%*)	12	0	0 (*0%*)	34	0
	2p-prop	0 (*0%*)	63	0	0 (*0%*)	12	0	0 (*0%*)	34	0
Gothic$_{\text{PROIEL}}$	rel-PoS	3 (*4.11%*)	73	0.03	–	–	–	0 (*0%*)	31	0
	1p-brackets	2 (*2.78%*)	72	0.02	–	–	–	0 (*0%*)	31	0
	2p-greedy	1 (*1.79%*)	56	0.01	2 (*13.33%*)	15	0.02	0 (*0%*)	31	0
	2p-prop	1 (*1.79%*)	56	0.01	2 (*13.33%*)	15	0.02	0 (*0%*)	31	0
Lithuanian$_{\text{HSE}}$	rel-PoS	2 (*4.17%*)	48	0.19	–	–	–	1 (*3.12%*)	32	0.09
	1p-brackets	7 (*15.91%*)	44	0.66	–	–	–	1 (*3.12%*)	32	0.09
	2p-greedy	3 (*8.11%*)	37	0.38	1 (*16.67%*)	6	0.09	1 (*3.12%*)	32	0.09
	2p-prop	3 (*8.11%*)	37	0.38	1 (*16.67%*)	6	0.09	1 (*3.12%*)	32	0.09

Table 16: Atomic label coverage of the encodings at testing time. *Unseen* denotes the number and percentage of unique labels not seen in the training and dev set with respect to the total number of unique labels appearing at the test time (*total*), while % *occ.* stands for occurrences of unseen labels with respect to the occurrences of all labels in the test set.

an increased label set size even though additional bracketing elements are introduced. For instance, in the most non-projective treebanks, such as Ancient Greek and Basque, the 2-planar encodings generate,

in fact, a smaller label space. This can be explained by the fact that the 2-planar encodings are likely to contain fewer unique sequences of bracketing symbols in general due to the distribution of bracketing elements among two planes.

Label coverage In sequence labeling parsing, the syntactic dependencies are encoded as a finite set of labels based on their presence in the training and development sets. Hence, it is anticipated that at the test time, some labels have not been seen during training. To explore to what extent it occurs, we examine the label coverage of the relative PoS-based, 1- and 2-planar bracketing-based encodings across various treebanks. In Table 16 we report the number of unseen labels from the training and development set, the total number of unique labels and the percentage of unseen label occurrences with respect to occurrences of all labels in the test set.

In general, it is discernible that only few labels are unknown at the test time out of all unique labels. For instance, in the case of 1P-BRACKETS in Ancient Greek, only 3.39% of unique labels (Task 1) were not seen during training. This translates into only four unique labels out of 118 that were unknown to the model at the test time. Moreover, when exploring how often they actually occur at the test time, they render solely 0.02% of the occurrences of all labels. Since the unseen labels are so rare, they presumably do not have a significant impact on the overall performance of a model.

4.4 Evaluation

In this section, we assess the relaxed 1-planar models in terms of UAS and LAS scores and their parsing speeds tested in different setups, followed by the results of the 2-planar variants. We extend the evaluation by comparing the bracketing-based family against the relative PoS-based encoding.

4.4.1 Performance of the relaxed 1-planar bracketing-based encoding

The aim of this experiment is to compare the performance of the encodings under various settings. Firstly, we leverage the fact that the bracketing-based encoding family does not require PoS tags, whose utility has been questioned in some previous work (Lhoneux et al., 2017; Anderson and Gómez-Rodríguez, 2020b). Hence, we will compare the performance of the relative PoS-based and the relaxed 1-planar bracketing-based encodings when obviating the need of PoS tags as input features. Secondly, their performance will be examined when relying on a different neural architecture. Finally, we evaluate the

sequence labeling models considering their speed and accuracy trade-off compared to some existing parsers.

Model We use two different sequence labeling encoders. The first one is based on BiLSTM within the NCRF++ framework as in the previous experiments, while the second one employs BERT (Devlin et al., 2019). More specifically, we use its pre-trained models [1]. For our multilingual setup we fine-tune the multilingual BERT (M-BERT:bert-base-multilingual-cased), unless language-specific models exist. An outline of our architecture using BERT as the encoder is illustrated in Figure 18.

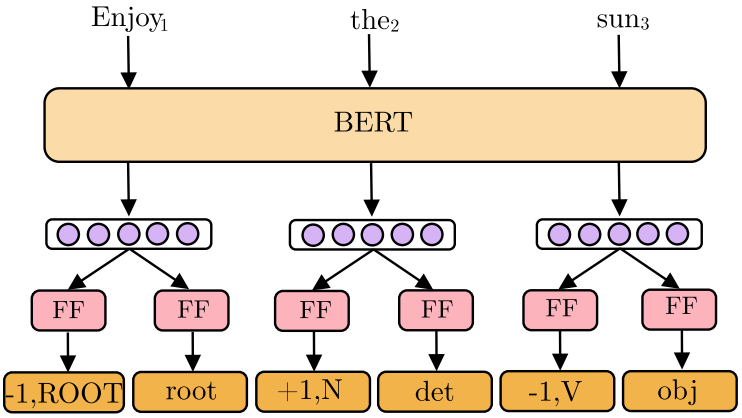


Figure 18: Sequence labeling model with BERT as encoder. We omit BERT's sub-word piece tokenization for simplicity.

Note that in BERT the input tokens are split into sub-words (WordPieces) (Wu et al., 2016) resulting in more sub-words than tokens in a given sentence. This impedes a direct assignment of a label to its corresponding token in our approach. To overcome it, we consider the first sub-word as a token. Moreover, unlike the BiLSTM-based model, BERT does not use PoS tags as input. Hence, we apply PoS tags only during decoding of the relative PoS-based labels.

We generate two-task labels for each word w_i by retrieving BERT's output hidden contextualized vectors h_i and using them as input to two separate feed-forward layers with *softmaxes*. Both models are optimized by computing the loss as the sum of the categorical cross-entropy of both tasks. The hyperparameters for the BiLSTM-based models are detailed in Section A.1.1 and for BERT in Section A.1.3.

Setup In this experiment unlike the previous one, the aim is to test the overall performance of the relaxed 1-planar encoding without focusing on non-projectivity. Hence, following Anderson and Gómez-Rodríguez (2020a) we select a representative subset of UDv2.4 (Nivre et al., 2019) treebanks: Ancient $\text{Greek}_{\text{Perseus}}$, $\text{Chinese}_{\text{GSD}}$, $\text{English}_{\text{EWT}}$,

1 https://huggingface.co/transformers/pretrained_models.html

$\text{Finnish}_{\text{TDT}}$, $\text{Hebrew}_{\text{HTB}}$, $\text{Russian}_{\text{GSD}}$, $\text{Tamil}_{\text{TTB}}$, $\text{Uyghur}_{\text{UDT}}$ and $\text{Wolof}_{\text{WTB}}$. Moreover, UDPipe (Straka and Straková, 2017) is used to obtain data with predicted segmentation, tokenization and PoS tags. In addition, each sentence is augmented with the special BOS and EOS symbols. For this setup, we use language-specific BERT models that are available for some of the selected languages, i.e. English (bert-base-cased), Chinese (bert-base-chinese) and Finnish (TurkuNLP/bert-base-finnish-cased-v1) (Virtanen et al., 2019).

As discussed previously, the utility of PoS tags in dependency parsing has been questioned. That is why we propose an experiment with two setups for the BiLSTM-based models. In the first one, denoted as PoS+, we proceed with our basic architecture, where the input vectors are a concatenation of the pre-trained word embeddings, char and PoS tag embeddings. In the second setup, in turn, we obviate the use of PoS tag vectors in the input and they will be only employed for decoding the labels of the relative PoS-based encoding. We will refer to this setup as PoS–. We consider this setup since in various circumstances it may not be desirable to use PoS tags as input features to the model. For instance, PoS tags may be not available for a given treebank, they may have a very low prediction accuracy or for a simple reason, some models do not use PoS tag features as in the case of BERT. Moreover, circumventing using PoS tags results in a faster pipeline and further simplicity of the model since there is no need of running a PoS tagger.

In the second experiment, we will test the relative PoS-based and the relaxed 1-planar bracketing-based encodings when relying on BERT as encoder, which by default does not use PoS tag vectors as input.

4.4.1.1 Impact of using PoS tags as input features

First, we examine the effect of using PoS tag embeddings as input to the BiLSTM-based models with the relative PoS-based and the relaxed 1-planar bracketing encoding. Table 17 shows the accuracy of the models in the two setups: (i) where PoS tag vectors are used as input (PoS+) or (ii) are left out (PoS–). In general, the results show that when the PoS tag vectors are used as input to the model (PoS+), the relative PoS-based encoding on average outperforms the bracketing-based one by 0.6% UAS on the test set, which confirms the preliminary results from the previous experiment on the English PTB treebank. The relative PoS-based models that use PoS tags as input features consistently achieve higher scores than the counterpart models across the selected languages, with the exception of Uyghur, where the bracketing-based model obtains higher accuracy both on the dev and test set. The largest discrepancy in the accuracy between the two encodings is prominent in Ancient Greek with an advantage of the head selection encoding of 4.6% UAS on the test set. The reason for the lower score of the re-

Split	Enc.	Ancient Greek		Chinese		English		Finnish		Hebrew		Russian		Tamil		Uyghur		Wolof		Avg	
		UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
dev_{PoS+}	rel-PoS	**70.2**	**61.8**	61.6	57.4	**82.6**	**79.0**	**83.2**	**79.3**	**67.4**	**63.8**	**84.0**	**79.6**	**59.5**	**52.0**	72.6	59.0	**77.0**	**70.5**	***73.1***	***66.9***
	1p-brackets	65.2	57.2	**61.8**	**57.9**	82.2	78.6	82.8	79.1	67.1	63.4	83.5	78.9	57.4	50.2	**73.1**	**59.9**	76.3	69.7	*72.2*	*66.1*
$test_{PoS+}$	rel-PoS	**69.5**	**60.0**	**64.4**	**60.0**	**82.0**	**78.5**	80.2	**74.9**	**63.0**	**59.5**	82.7	77.3	**58.0**	**49.8**	71.5	57.8	**76.1**	**69.6**	***71.9***	***65.3***
	1p-brackets	64.9	56.2	64.2	59.8	81.7	78.3	**81.1**	76.0	62.1	58.7	**82.9**	**77.9**	57.2	49.0	**72.0**	**58.5**	75.4	68.8	*71.3*	*64.8*
dev_{PoS-}	rel-PoS	**65.3**	**58.3**	58.8	55.3	80.3	77.1	80.8	77.3	65.3	62.2	81.3	77.4	52.7	41.9	65.7	53.3	73.1	66.3	*69.2*	*63.2*
	1p-brackets	64.7	57.2	**63.8**	**59.4**	**83.4**	**80.0**	**84.1**	**80.4**	**68.7**	**65.0**	**84.0**	**79.7**	**55.9**	**45.1**	**72.1**	**58.8**	**75.2**	**67.4**	***72.4***	***65.9***
$test_{PoS-}$	rel-PoS	62.9	**55.1**	60.3	56.8	78.5	75.3	65.6	59.6	59.7	56.6	79.0	73.8	51.6	40.6	64.9	52.4	72.3	65.6	*66.1*	*59.5*
	1p-brackets	**63.4**	54.8	**65.3**	**61.1**	**82.4**	**78.9**	**75.1**	**67.7**	**62.3**	**58.6**	**81.8**	**75.8**	**56.9**	**42.6**	**71.0**	**57.4**	**75.1**	**67.2**	***70.4***	***62.7***

Table 17: Results for the BiLSTM-based models, in which PoS tags are used as input features (PoS+) or are omitted (PoS–).

laxed 1-planar bracketing-based model is likely due to the high degree of non-projective sentences present in that treebank.

When analyzing the accuracy of the models without relying on PoS tags as input (PoS-), the tendency is reversed. The bracketing-based models outperform the counterpart models on average by 4.3% UAS on the test set demonstrating the strong dependency of the relative PoS-based encoding on the input PoS tag vectors. Hence, by any reason PoS tags are not used as input features, the bracketing-based encoding may be a favorable option.

4.4.1.2 Impact of using a different architecture

Now, we move on to examine the performance of the encodings when taking into account a different neural architecture. Specifically, we rely on a model with BERT as encoder that does not use PoS tags as input features. Table 18 compares the encodings for the selected languages with the available pre-trained models. The results show that the 1-planar bracketing-based models clearly outperform the relative PoS-based ones across all languages. On average, the bracketing-based encoding has an advantage of 5.4% UAS and 4.3% LAS on the test set. This can be partly explained by the fact that BERT does not use PoS tag features as input, hence we can observe a similar tendency as in the BiLSTM models in the PoS– setup. The models that rely on a language-specific pre-trained BERT, such as Chinese, English and Finnish obtain overall higher accuracy than the BiLSTM models (PoS–). Conversely, the models based on M-BERT such as Hebrew, Russian and Tamil achieve lower accuracy when compared to the BiLSTM-based ones. As observed in Wu and Dredze (2020), it is not guaranteed that languages supported with M-BERT perform better than when learned with models that do not rely on pretraining. This is especially prominent in low-resource scenarios.

Split	Encoding	Chinese$_{GSD}$		English$_{EWT}$		Finnish$_{TDT}$		Hebrew$_{HTB}$		Russian$_{GSD}$		Tamil$_{TTB}$		Avg	
		UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
dev	rel-PoS	59.8	57.0	82.3	79.9	82.6	79.7	61.8	58.0	77.6	73.2	43.4	33.6	*67.9*	*63.5*
	1p-brackets	**69.3**	**65.3**	**87.2**	**84.4**	**88.6**	**85.1**	**66.6**	**61.7**	**80.6**	**75.4**	**49.8**	**36.8**	***73.7***	***68.1***
test	rel-PoS	61.5	58.8	81.0	78.8	82.6	80.0	56.9	53.1	77.9	73.2	45.2	34.1	*67.5*	*63.0*
	1p-brackets	**70.2**	**66.5**	**86.2**	**83.3**	**89.4**	**86.1**	**62.7**	**57.9**	**79.9**	**74.2**	**48.9**	**35.9**	***72.9***	***67.3***

Table 18: Performance comparison of the relative PoS-based and the relaxed 1-planar bracketing-based encoding using BERT on the languages with available pre-trained models.

4.4.1.3 Encoding evaluation with gold data

In the previous experiment in the PoS+ setup, the models were tested on the input sentences with predicted segmentation, tokenization and PoS tags. In order to gain insight into the impact of the predicted setup on the encodings and to examine whether some encoding may suffer more from this setup, we test the BiLSTM-based models on the gold data. Table 19 shows the upper bound for the relative PoS-based and relaxed 1-planar bracketing-based encodings that can be obtained on the gold data. We provide the percentage increase in parenthesis with respect to the results from Table 17.

Language	rel-PoS		1p-brackets	
	UAS	LAS	UAS	LAS
Ancient Greek$_{Perseus}$	**74.8** *(+7.6%)*	**68.3** *(+13.9%)*	69.1 *(+6.4%)*	63.1 *(+12.3%)*
Chinese$_{GSD}$	**84.9** *(+31.8%)*	**82.0** *(+36.7%)*	84.2 *(+31.1%)*	81.2 *(+35.8%)*
English$_{EWT}$	**89.3** *(+8.9%)*	**86.9** *(+10.7%)*	88.8 *(+8.7%)*	86.5 *(+10.4%)*
Finnish$_{TDT}$	85.1 *(+6.2%)*	81.3 *(+8.5%)*	**86.1** *(+6.2%)*	**82.5** *(+8.5%)*
Hebrew$_{HTB}$	**88.2** *(+40.1%)*	**85.4** *(+43.7%)*	87.5 *(+40.9%)*	84.7 *(+44.3%)*
Russian$_{GSD}$	**86.3** *(+4.3%)*	81.5 *(+5.3%)*	86.0 *(+3.7%)*	**81.6** *(+4.8%)*
Tamil$_{TTB}$	**76.5** *(+31.9%)*	**68.7** *(+37.9%)*	73.6 *(+28.5%)*	66.8 *(+36.3%)*
Uyghur$_{UDT}$	74.3 *(+4.0%)*	61.1 *(+5.7%)*	**74.7** *(+3.7%)*	**61.7** *(+5.5%)*
Wolof$_{WTB}$	**86.0** *(+13.0%)*	**81.8** *(+17.5%)*	85.3 *(+13.1%)*	80.7 *(+17.2%)*

Table 19: Performance of the BiLSTM-based models in the PoS+ setup evaluated on the test set of UD treebanks with gold segmentation, tokenization and PoS tags. In parentheses the percentage increase is given with respect to the results obtained with the predicted setup.

The results show that the prediction accuracy of the upstream tasks has a relatively large impact on the models. The greatest gap between the results on the predicted and gold data is noticeable in Hebrew. In this case, using the gold data results in a percentage increase of 40.1% of UAS (rel-PoS) and 40.9% (1p-brackets) followed by Chinese (31.8% and 31.1%, respectively) and Tamil (31.9% and 28.5%). On average, the relative PoS-based models have a higher percentage increase on the gold data set suggesting that these models get penalized more due to the predicted PoS. This can be explained by their strong reliance on PoS

tags whose prediction accuracy may vary. For instance, in Chinese the advantage of PoS-based encoding in UAS with respect to the bracketing-based one decreases from +0.7 on the gold data set to +0.2 on the predicted data set. A similar inclination can be observed to a greater extent in the remaining languages.

4.4.1.4 Comparison of parsers' speed and accuracy trade-off

Since one of the key aspects of our approach is to provide a fast and accurate parser, we will now compare the relative PoS-based and the relaxed 1-planar bracketing-based models against some existing SOTA dependency parsers. In particular, we will contrast the systems with respect to the speed and accuracy trade-off they maintain. To do so, we select four sequence labeling models based on the Pareto frontier from Section 4.3.1.1. More specifically, the models are picked across the span of the frontier: the most accurate one (rel-PoS$^{\text{C}}_{2,800}$), the fastest one (1p-brackets$_{2,250}$) and two that fall in the center (rel-PoS$_{2,250}$, rel-PoS$^{\text{C}}_{2,400}$). Table 20 compares the accuracy and speed of the selected models against some existing dependency parsers.

Model	sent/s		UAS	LAS
	CPU	GPU		
Kiperwasser and Goldberg (2016) (T)	$76_{\pm 1}$	–	93.90	91.90
Kiperwasser and Goldberg (2016) (G)	$80_{\pm 0}$	–	93.10	91.00
Chen and Manning (2014)	654†	–	91.80	89.60
Dozat and Manning (2017)	–	411†	95.74	94.08
Ma et al. (2018)	–	$10_{\pm 0}$	95.87	94.19
rel-PoS$_{2,250}$	$267_{\pm 1}$	$777_{\pm 24}$	92.95	90.96
rel-PoS$^{\text{C}}_{2,400}$	$165_{\pm 1}$	$700_{\pm 5}$	93.34	91.34
rel-PoS$^{\text{C}}_{2,800}$	$101_{\pm 2}$	$648_{\pm 20}$	**93.67**	**91.72**
1p-brackets$_{2,250}$	$310_{\pm 30}$	$730_{\pm 53}$	92.64	90.59

Table 20: Comparison of models on the PTB test set. The symbol † indicates that the speed comes from the original papers.

In general, the results show that our sequence labeling parsers yield competitive performance and preserve a good speed/accuracy trade-off. The most accurate model from the Pareto frontier, rel-PoS$^{\text{C}}_{2,800}$, is nearly on par (−0.18 LAS) with the accuracy of the transition-based BIST parser (Kiperwasser and Goldberg, 2016) while being faster. Despite the fact that both parsers rely on the BiLSTM representation, the sequence labeling parser characterizes its simplicity: it does not depend on any transition system and circumvents the need of a stack during training and parsing or a dynamic oracle. The model rel-PoS$_{2,250}$ which uses BIST hyperparameters is 3.34x faster than the graph-based BIST

at minimal cost of accuracy (−0.04 LAS). It is also 3.51x faster at the cost of −0.94 LAS score compared to the transition-based BIST parser. 1p-brackets$_{2,250}$ is the fastest among the sequence labeling models, however at some expense of accuracy.

4.4.2 Performance of the 2-planar bracketing-based encodings

Next, we evaluate the performance of the 2-planar bracketing-based encodings in terms of UAS and LAS compared to the other encodings. Table 21 reports the results of the models on the development and test sets from a range of highly non-projective treebanks including the control treebanks. In this evaluation, the predicted PoS tags are not used as input to the model. The reason for it is that we are interested in testing the performance of the models in a setup, where PoS tags are not available for the reasons we discussed earlier. Finally, we will also explore whether adding a second plane has an impact on the parsing speed.

Language	Encoding	dev		test		Language	Encoding	dev		test	
		UAS	LAS	UAS	LAS			UAS	LAS	UAS	LAS
Ancient Greek$_{\text{Perseus}}$	rel-PoS	65.29	58.27	62.91	55.07	Korean$_{\text{Kaist}}$	rel-PoS	81.47	78.50	77.25	73.92
	1p-brackets	64.70	57.21	63.36	54.80		1p-brackets	84.54	81.54	**82.37**	**79.03**
	2p-greedy	67.10	59.97	**65.90**	**57.15**		2p-greedy	85.01	82.01	82.33	78.91
	2p-prop	67.06	59.84	65.11	56.55		2p-prop	84.65	81.73	82.32	**79.03**
Basque$_{\text{BDT}}$	rel-PoS	77.48	72.91	75.28	70.19	Danish$_{\text{DDT}}$	rel-PoS	78.28	74.93	77.07	73.45
	1p-brackets	80.13	75.37	78.37	72.95		1p-brackets	80.60	76.59	78.25	73.94
	2p-greedy	79.98	75.18	78.13	72.63		2p-greedy	80.68	76.80	78.49	74.07
	2p-prop	80.44	75.56	**78.58**	**73.08**		2p-prop	81.15	77.27	**78.87**	**74.42**
Hungarian$_{\text{Szeged}}$	rel-PoS	72.58	67.13	66.19	59.32	Gothic$_{\text{PROIEL}}$	rel-PoS	65.25	58.58	**67.14**	**59.72**
	1p-brackets	75.09	69.13	67.80	60.50		1p-brackets	65.26	57.92	66.63	59.02
	2p-greedy	75.47	69.33	**68.07**	**60.74**		2p-greedy	65.26	58.05	66.84	59.26
	2p-prop	75.26	69.05	67.95	60.63		2p-prop	65.29	57.91	66.25	58.41
Portuguese$_{\text{Bosque}}$	rel-PoS	87.10	84.28	84.74	81.02	Lithuanian$_{\text{HSE}}$	rel-PoS	39.04	26.37	31.05	19.70
	1p-brackets	88.88	85.78	**86.67**	**82.44**		1p-brackets	40.97	25.63	34.62	19.42
	2p-greedy	88.88	85.76	86.51	82.39		2p-greedy	41.34	26.46	**35.08**	20.45
	2p-prop	89.00	85.82	86.52	82.17		2p-prop	44.19	29.03	34.80	**21.29**
Urdu$_{\text{UDTB}}$	rel-PoS	80.98	75.09	81.18	75.26	*Japanese$_{GSD}$*	*rel-PoS*	*76.60*	*75.83*	*74.83*	*73.96*
	1p-brackets	84.22	77.23	**84.28**	77.19		*1p-brackets*	*78.73*	*77.67*	*77.34*	*76.10*
	2p-greedy	84.01	77.16	84.08	77.19		*2p-greedy*	*78.81*	*77.78*	*77.47*	*76.24*
	2p-prop	83.89	77.30	84.26	**77.41**		*2p-prop*	*78.81*	*77.78*	*77.47*	*76.24*
Afrikaans$_{\text{AfriBooms}}$	rel-PoS	79.00	74.58	78.93	74.65	*Galician$_{CTG}$*	*rel-PoS*	*79.72*	*76.40*	*78.36*	*75.05*
	1p-brackets	80.77	75.54	79.52	74.86		*1p-brackets*	*80.82*	*77.35*	*80.02*	*76.33*
	2p-greedy	81.41	76.33	**80.13**	**75.53**		*2p-greedy*	*80.90*	*77.36*	*79.91*	*76.32*
	2p-prop	81.50	76.30	79.96	75.43		*2p-prop*	*80.90*	*77.36*	*79.91*	*76.32*

Table 21: Comparison of UAS and LAS (%) obtained by the models relying on the relative PoS-based, 1- and 2-planar bracketing-based encodings and evaluated on the predicted dev and test set of highly non-projective treebanks. Japanese and Galician treebanks are fully projective and serve as the control treebanks.

The results show that the 2-planar bracketing-based encodings outperform the 1-planar counterpart in the majority of treebanks. The

improvements vary across the languages but on average 2P-GREEDY increases UAS by 0.4%, while 2P-PROP by 0.3% on the test set. Both encodings improve LAS by 0.4%. However, when taking into consideration the most non-projective treebank, Ancient Greek, 2P-GREEDY outperforms 2P-PROP by 0.79% UAS and 0.6% LAS. Hence again, the theoretical advantage in arc coverage of 2P-PROP does not seem to be fully reflected in terms of UAS and LAS. The results suggest that the greedy plane assignment strategy is presumably easier to learn by the model.

4.4.2.1 Parsing speed

Moreover, we measure parsing speeds of the 2-planar bracketing-based models and compare them to the speeds of their counterparts. As mentioned earlier, enforcing dependency parsers to handle non-projectivity has often led to increased computational cost. Hence, we examine how much the extended output vocabulary of the 2-planar encodings impacts the efficiency of our sequence labeling parser. The parsing speeds are reported in Table 22[2].

The results suggest that the parsing speed does not significantly differ between the 1- and 2-planar models. More specifically, the computation of an additional *softmax* to represent the second plane does not seem to contribute to an extra computational cost, even if it requires handling an additional output vocabulary. Moreover, it should be noted that the relative PoS-based encoding requires PoS tags in order to decode its labels into the dependency trees. This implies that its total parsing time needs to be augmented with the computation time of a PoS tagging step. For orientation purposes, in Table 23 we provide exemplary prediction times of pre-trained UDPipe models for PoS tagging.

4.5 Strengths and limitations

When drawing comparisons at the level of the encoding families, the bracketing-based encodings perform slightly worse than the relative PoS-based one when PoS tags are used as input features to the model. However, one of the assets of the bracketing-based encoding family is that they do not depend on any features, unlike the relative PoS-based encoding. Hence, it can be an alternative when the use of PoS tags is not preferable, for instance, when their quality or frequency is low, they are not available for a given language or one does not desire to include a PoS tagger in the pipeline. Furthermore, some architectures may simply not use PoS tag embeddings as input. In fact, when PoS tags are not used as input features, the bracketing-based models consid-

2 The parsing speeds are measured in sentences per second on a single core of a CPU (Intel Core i7-8700 CPU 3.2 GHz) and on a GPU (Nvidia TITAN Xp).

Language	Encoding	sent/s CPU	sent/s GPU	Language	Encoding	sent/s CPU	sent/s GPU
Ancient Greek$_{\text{Perseus}}$	rel-PoS†	305	1012	Korean$_{\text{Kaist}}$	rel-PoS†	442	1718
	1p-brackets	303	1011		1p-brackets	447	1718
	2p-greedy	288	889		2p-greedy	434	1598
	2p-prop	289	886		2p-prop	435	1544
Basque$_{\text{BDT}}$	rel-PoS†	387	1461	Danish$_{\text{DDT}}$	rel-PoS†	279	962
	1p-brackets	388	1454		1p-brackets	280	980
	2p-greedy	378	1369		2p-greedy	265	841
	2p-prop	378	1369		2p-prop	264	832
Hungarian$_{\text{Szeged}}$	rel-PoS†	219	802	Gothic$_{\text{PROIEL}}$	rel-PoS†	430	1266
	1p-brackets	221	797		1p-brackets	429	1269
	2p-greedy	212	739		2p-greedy	414	1175
	2p-prop	213	750		2p-prop	412	1186
Portuguese$_{\text{Bosque}}$	rel-PoS†	242	868	Lithuanian$_{\text{HSE}}$	rel-PoS†	239	828
	1p-brackets	246	872		1p-brackets	235	769
	2p-greedy	236	811		2p-greedy	228	740
	2p-prop	237	814		2p-prop	229	730
Urdu$_{\text{UDTB}}$	rel-PoS†	182	625	*Japanese$_{GSD}$*	*rel-PoS†*	*214*	*663*
	1p-brackets	186	616		*1p-brackets*	*214*	*661*
	2p-greedy	174	544		*2p-greedy*	*206*	*611*
	2p-prop	175	549		*2p-prop*	*205*	*616*
Afrikaans$_{\text{AfriBooms}}$	rel-PoS†	228	861	*Galician$_{CTG}$*	*rel-PoS†*	*175*	*752*
	1p-brackets	228	857		*1p-brackets*	*177*	*756*
	2p-greedy	220	805		*2p-greedy*	*170*	*673*
	2p-prop	221	805		*2p-prop*	*170*	*673*

Table 22: Comparison of parsing speeds (sent/s) on the test sets averaged over 5 runs and measured on a single core CPU and a GPU. Parsing speeds with a symbol † denote that the encoding requires an additional step of computing PoS tags whose time needs to be added.

Language	sent/s	Language	sent/s
Ancient Greek$_{\text{Perseus}}$	567	Korean$_{\text{Kaist}}$	921
Basque$_{\text{BDT}}$	881	Danish$_{\text{DDT}}$	671
Hungarian$_{\text{Szeged}}$	755	Gothic$_{\text{PROIEL}}$	861
Portuguese$_{\text{Bosque}}$	210	Lithuanian$_{\text{HSE}}$	1418
Urdu$_{\text{UDTB}}$	45	Japanese$_{\text{GSD}}$	767
Afrikaans$_{\text{AfriBooms}}$	465	Galician$_{\text{CTG}}$	366

Table 23: UDPipe tagging speed (in sent/s) on the test sets measured on a single core CPU.

erably outperform the relative PoS-based encoding in most languages. Moreover, they have an advantage of generating a more compact label vocabulary which may ease the label learning.

One of the downsides of the relaxed 1-planar bracketing-based encoding is that unlike the PoS-based counterpart, it only partially supports non-projectivity. More specifically, it is able to solely reconstruct non-projective arcs in the opposite direction, while arcs in the same direction are transformed into projective ones.

To mitigate this limitation and reduce the gap to the relative PoS-based encoding, we proposed 2-planar variants. In general, models that rely on the 2-planar encodings with greedy plane assignment and with restriction propagation provide almost fully coverage of non-projective arcs that directly translates into an improved accuracy. The 2-planar models outperform the relaxed 1-planar models across multiple highly non-projective treebanks without sacrificing efficiency yielding a useful alternative to the relative PoS-based encoding.

4.6 Conclusions

In this chapter, we have introduced a new family of encodings that represents dependency arcs in terms of balanced bracketing elements with directional awareness. First, we have presented the relaxed 1-planar bracketing-based encoding and discussed its limitation with respect to the coverage of non-projectivity, which was the main motivation for introducing variants that rely on the property of 2-planarity. Then, we have described how 2-planarity can be applied to sequence labeling parsing. On these grounds, we have proposed two encodings with different plane assignment strategies. The first strategy assigns an arc to a plane greedily. More specifically, a crossing arc is placed on a plane if no other crossing arcs are already assigned to that plane. The second strategy, in turn, assigns a plane based on restriction propagation on the crossings graph. More specifically, when assigning an arc to a given plane, all crossing arcs are forbidden from that plane and the neighbors of the neighbors in the crossings graph are forbidden from the other plane. Therefore, the latter encoding guarantees a full coverage of 2-planar trees.

Thereafter, we have tested empirically all encodings and compared them to the best encoding from the head selection family introduced in Chapter 3. We have first conducted a set of in-depth analyses focusing on different assets of the relaxed 1-planar and 2-planar bracketing encodings. We have compared efficiency of the sequence labeling models based on both encoding families that varied in their hyperparameter combinations. Moreover, we have demonstrated that the bracketing-based encodings provide a more compact label representation in terms of label set size compared to the relative PoS-based encoding. We have also tested the coverage of non-projectivity among the 2-planar variants and shown that they almost fully handle crossing arcs, and thereby obviating the limitation with respect to the head selection encodings that are able to fully encode non-projectivity.

Secondly, we have provided an evaluation of the encodings and have compared the performance of the relaxed 1-planar bracketing-based and the relative PoS-based encodings in terms of accuracy with respect to the impact of PoS tags as input features and the use of a different architecture. The results have shown that the relative PoS-based

encoding outperforms the relaxed 1-planar bracketing-based encoding only when PoS tags are used as input features. The tendency is reversed when PoS tags are not applied to the model, as well as when BERT is used. Furthermore, we have shown that both the 1-planar bracketing-based and the relative PoS-based models provide a good speed/accuracy trade-off compared to some existing parsers, proving that the sequence labeling parsers are efficient. Regarding the 2-planar bracketing-based encodings, their almost full coverage of non-projective trees is reflected in the improved accuracy with respect to the relaxed 1-planar encoding in highly non-projective treebanks. Furthermore, extending the bracketing-based encoding to handle non-projectivity has been shown to come at almost no additional computational cost.

5 Transition-based encodings

In this chapter, we will introduce a set of encodings that rely on the left-to-right transition-based systems. In order to obtain such encodings, we propose a mapping method for representing a sequence of transitions as a sequence of labels (one per word), where each label corresponds to a subsequence of transitions. The contribution is twofold: firstly, this method enables establishing a theoretical link between the transition-based and sequence labeling parsing paradigms. Secondly, it facilitates an automatic derivation of new encodings by exploiting the existing left-to-right systems that can be applied to a fast and simple sequence labeling parsing.

We start with defining a generic mapping method applicable to a range of left-to-right systems, followed by a description of the decoding process of transition-based labels. Furthermore, we will examine our approach empirically in terms of accuracy and we will compare and analyze the facets of encodings derived from different transition-based systems. Finally, we will discuss the strengths and limitations of this approach.

5.1 Transition-based systems

A transition system can be represented as an abstract automaton with finite states linked by transitions, where each state holds a *configuration* (c) that usually consists of a *stack* (σ), *buffer* (β) and a set of dependency arcs (A). In order to learn to move from one state to another, such a system is trained to approximate an *oracle* which is a function that maps configurations to transitions. The selected transition is then applied resulting in a new configuration. Hence, a parse of a sentence w can be obtained by iterating through the states, while consecutively applying transitions chosen by the oracle. The path starts at an initial configuration and eventually terminates in a final configuration with an empty buffer. This succession of states is called a *transition sequence*.

More formally, following Nivre (2008) a *transition system* for an input sentence $w = w_1, w_2, ..., w_n$ can be defined as a quadruple $S = (C, T, c_s, C_t)$, where:

- C is a set of configurations
- T is a set of transitions, where $t : C \rightarrow C$

- c_s is an initialization function that maps the input w to a unique initial configuration, $c_s(w) \in C$
- $C_t \subseteq C$ is a set of final configurations

Transition systems
Initial config: $c_s(w_1 \dots w_n)$: $([root], [1...n], \emptyset)$
Final config: $C_t = \{([], [], A)\}$
ARC-STANDARD
SH $(\sigma, b\|\beta, A) \Rightarrow (\sigma\|b, \beta, A)$
LA$_{st}$ $(\sigma\|s_1\|s_0, \beta, A) \Rightarrow (\sigma\|s_0, \beta, A \cup \{s_0 \rightarrow s_1\})$
RA$_{st}$ $(\sigma\|s_1\|s_0, \beta, A) \Rightarrow (\sigma\|s_1, \beta, A \cup \{s_1 \rightarrow s_0\})$
ARC-EAGER
SH $(\sigma, b\|\beta, A) \Rightarrow (\sigma\|b, \beta, A)$
LA$_e$ $(\sigma\|s, b\|\beta, A) \Rightarrow (\sigma, b\|\beta, A \cup \{b \rightarrow s\})$
if $\nexists k : k \rightarrow s \in A$
RA$_e$ $(\sigma\|s, b\|\beta, A) \Rightarrow (\sigma\|s\|b, \beta, A \cup \{s \rightarrow b\})$
REDUCE $(\sigma\|s, \beta, A) \Rightarrow (\sigma, \beta, A)$
if $\exists k : k \rightarrow s \in A$
ARC-HYBRID
SH $(\sigma, b\|\beta, A) \Rightarrow (\sigma\|b, \beta, A)$
LA$_e$ $(\sigma\|s, b\|\beta, A) \Rightarrow (\sigma, b\|\beta, A \cup \{b \rightarrow s\})$
RA$_{st}$ $(\sigma\|s_1\|s_0, \beta, A) \Rightarrow (\sigma\|s_1, \beta, A \cup \{s_1 \rightarrow s_0\})$
COVINGTON (NON-PROJECTIVE)
Initial config: $c_s(w_1 \dots w_n)$: $([], [], [root, 1...n], \emptyset)$
Final config: $C_t = \{(\lambda_1, \lambda_2, [], A)\}$
SH$_c$ $(\lambda_1, \lambda_2, b\|\beta, A) \Rightarrow (\lambda_1 \cdot \lambda_2\|b, [], \beta, A)$
NO-ARC$_c$ $(\lambda_1\|s, \lambda_2, \beta, A) \Rightarrow (\lambda_1, s\|\lambda_2, \beta, A)$
LA$_c$ $(\lambda_1\|s, \lambda_2, b\|\beta, A) \Rightarrow (\lambda_1, s\|\lambda_2, b\|\beta, A \cup \{b \rightarrow s\})$
if $\nexists k : k \rightarrow s \in A$ (single head)
if $\nexists$ path $s \rightarrow ... \rightarrow b$ in A (acyclicity)
RA$_c$ $(\lambda_1\|s, \lambda_2, b\|\beta, A) \Rightarrow (\lambda_1, s\|\lambda_2, b\|\beta, A \cup \{s \rightarrow b\})$
if $\nexists k : k \rightarrow b \in A$ (single head)
if $\nexists$ path $b \rightarrow ... \rightarrow s$ in A (acyclicity)

Table 24: Transition operators and configurations in the selected transition-based systems.

Throughout the years, a wide variety of transition systems have been proposed. Some of the well-established systems are based on arc-standard (Fraser, 1989; Nivre, 2004) or arc-eager algorithms (Nivre, 2004) that served as the basis for a manifold of variants, i.a. arc-hybrid (Kuhlmann, Gómez-Rodríguez, and Satta, 2011), arc-eager with buffer transitions (Fernández-González and Gómez-Rodríguez, 2012) or

spinal arc-eager (Ballesteros and Carreras, 2015). Moreover, various systems were developed to handle non-projectivity (Covington, 2001; Nivre, 2009; Gómez-Rodríguez and Nivre, 2010) or were based on alternative algorithms (Goldberg and Elhadad, 2010; Sartorio, Satta, and Nivre, 2013). Since there is a broad spectrum of other transition-based parsers, we will include a more detailed overview of transition systems later on in this chapter.

In this work, we primarily focus on four algorithms that, as we will show in Section 5.2, are *left-to-right* systems and hence can be applied in sequence labeling parsers. More concretely, we will use encodings retrieved from ARC-STANDARD (Fraser, 1989; Nivre, 2004), ARC-EAGER (Nivre, 2004) and ARC-HYBRID (Kuhlmann, Gómez-Rodríguez, and Satta, 2011) systems that are fully projective and fall into the group of stack-based algorithms (Nivre, 2008), as well as a non-projective COVINGTON (Covington, 2001) that is a list-based algorithm. In general, a stack-based configuration is defined as a triple $c = (\sigma, \beta, A)$, where the element σ denotes a stack that holds partially processed tokens, β is a buffer with the remaining tokens and A is a set of already created arcs. The non-projective Covington, in turn, operates on list-based configurations defined as $c = (\lambda_1, \lambda_2, \beta, A)$, where λ_1 and λ_2 are lists with partially processed tokens. The transition operators and configurations of the selected systems are detailed in Table 24.

Generally, the stack-based systems differ in how transition operators are defined and how an action modifies the stack and the buffer. Additionally, they also may have different preconditions. For instance, the main distinction between the arc-standard and arc-eager algorithms is that the latter has an additional transition operator REDUCE and the right arc is attempted to be assigned earlier (more eagerly) than in the arc-standard algorithm. The arc-hybrid system, in turn, is endowed with a combination of the LEFT-ARC defined as in arc-eager and RIGHT-ARC as in arc-standard. Moreover, it is also worth noting that among the algorithms used in this work, solely the variant of the Covington system is able to handle non-projectivity.

Now, we will define how transition-based systems can be mapped to sequence labeling parsers using read transitions.

5.2 Mapping transition-based parsers to sequence labeling parsers

In order to apply transitions in a sequence labeling parser, we need to find a method for mapping an arbitrary transition sequence generated by a transition-based system for a sentence of length n to a sequence of n labels (one per word). To do so, we identify *read transitions* as those that read a new token from the input in a left-to-right fashion. More formally, we define a system $S = (C, T, c_s, C_t)$ as a *left-to-right*

transition system, if there are read transitions which form a subset of transitions $T_s \subseteq T$ and which satisfy the following two conditions:

1. There are exactly n read transitions that correspond to a sequence of transitions $t_1, \ldots, t_m$ for a sentence $w_1 \ldots w_n$, where t_1 needs to be one of the read transitions.

2. There is a constant value k such that, for each $1 \leq i \leq n$, computations with i read transitions result in a partial parse restricted to $w_1 \ldots w_{i+k}$.

This implies that processing an input sentence with n tokens will require n read transitions. A transition sequence for a given sentence can be then split using read transitions, such that each of the subsequences can be assigned to a label for a given token w_i. Thus, *every* complete computation is of this form:

$$t_1^r, t_1^1, \ldots, t_1^{m_1}, t_2^r, t_2^1, \ldots, t_2^{m_2}, \ldots, t_n^r, t_n^1, \ldots, t_n^{m_n}$$

t_i^r denotes a read transition followed by non-read transitions t_i^j. Thereby, each ith label is assigned the corresponding subsequence of transitions $t_i^r, t_i^1, \ldots, t_i^{m_i}$ starting with the reading action t_i^r.

More concretely, in the case of the arc-standard, arc-hybrid and the non-projective Covington algorithms, a transition sequence is split on the read transition SHIFT that reads tokens from the buffer. Figure 19 illustrates an example of such a split, where each SHIFT initiates a new label span holding a subsequence of transitions.

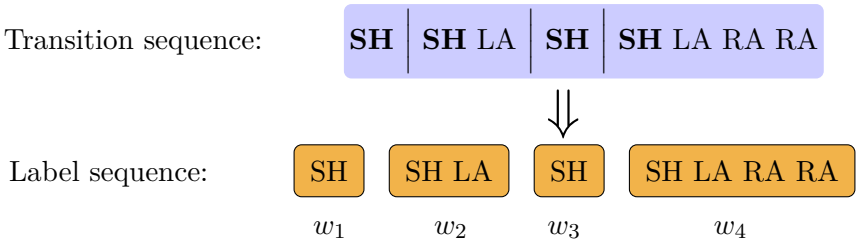


Figure 19: An example of a transition sequence corresponding to a sentence of length n that is split into a sequence of n labels using the read transition SH (in bold). Vertical lines denote the label boundaries starting with the read transition.

In the case of the arc-eager system, it has an additional read transition: the right-arc, since it also shifts a token from the buffer to the stack. As an example, Figure 20 compares how labels with transition subsequences are distributed using different transition systems.

Regarding the value of the constant k, in a transition system such as arc-standard k=0, since words need to be placed on the stack in order to create dependencies between them. Hence, with i read transitions only words $w_1 \ldots w_i$ are considered in the partial parse. In the example from Figure 20, the arc between *Kyrie*$_1$ and *ate*$_2$ can be first created after

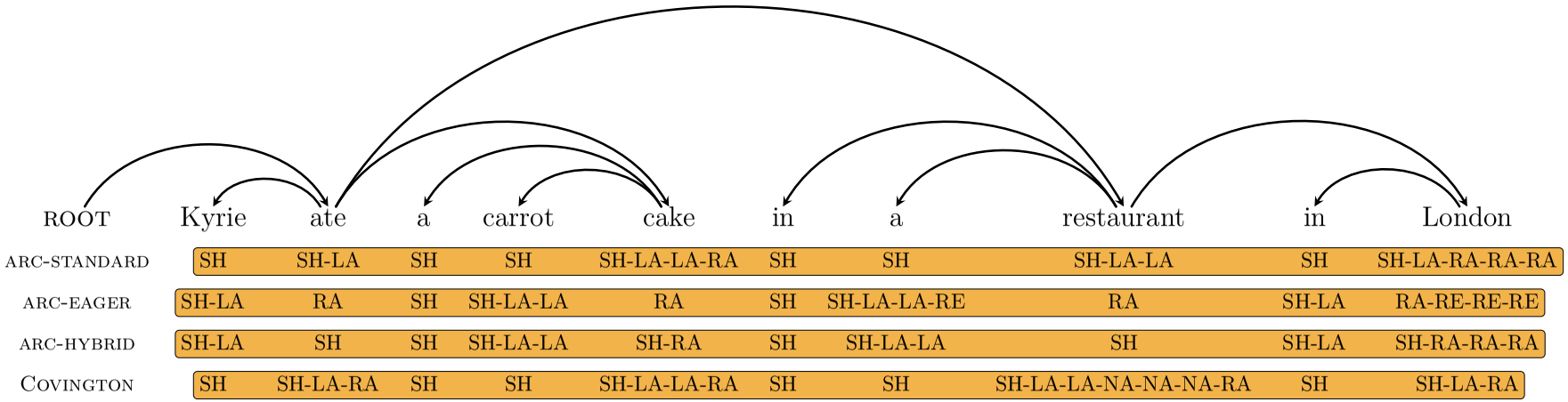


Figure 20: Label distribution for transition sequences derived from various transition-based systems. LA denotes Left-Arc, RA: Right-Arc, SH: Shift and NA: No-Arc.

two read transitions (SH). In other words, the value of the constant k is considered as the offset between the number of the applied read transitions and the number of words that can be parsed in a given state. Contrarily, in a transition system such as arc-eager k=1, as the dependencies can be also created by involving the first word in the buffer. Hence, the words considered for parsing are $w_1 \dots w_{i+k}$. As in the example, the token $Kyrie_1$ can be assigned as a dependent of ate_2 after a single read transition. Moreover, since the transition sequence in the non-projective Covington algorithm always terminates with a shift action, we omit it in the representation of the last label.

In Table 25, we provide an overview of some existing transition-based parsers that we classify with respect to whether they are left-to-right, what type of read transitions they apply and the value of the constant k. We show that our mapping can be applied to the majority of the well-known left-to-right[1] transition-based systems and hence they can be successfully adapted to sequence labeling parsing.

5.3 Decoding

During the decoding process, we start by reading each label corresponding to the input token in a sentence. It is anticipated that some labels may contain illegal actions that will violate some preconditions of a certain transition system. For instance, a model may wrongly predict LEFT-ARC in the arc-standard algorithm, when the second top element on the stack is the syntactic ROOT. Similarly, LEFT-ARC may be incorrectly predicted in the arc-eager algorithm, when the top element in the stack already has a head. These actions are forbidden, hence in such cases we discard them and proceed with the next action in the sequence.

One of the assets of the transition-based encodings is that they guarantee acyclicity and single head constraint. However, since all illegal actions are discarded due to the precondition violation, some words may not be assigned any head. To recover from it, we apply the common postprocessing steps as described in Section 3.2.1. Postprocessing, in fact, has been recently a tendency in some SOTA parsers that first generate ill-formed trees that are afterwards postprocessed. For instance, Dozat and Manning (2017) only ensure a valid output tree at testing time by applying the MST algorithm to the output of their graph-based parser.

1 It is worth noting that we consider the left-to-right incrementality at the transition system level.

Algorithm	L2R?	Read t.	k
Arc-standard (Fraser, 1989; Nivre, 2004)	Yes	SH	0
Arc-eager (Nivre, 2003)	Yes	SH, RA	1
Arc-hybrid (Kuhlmann, Gómez-Rodríguez, and Satta, 2011)	Yes	SH	1
Covington projective (Covington, 2001; Nivre, 2008)	Yes	SH	0
Covington non-projective (Covington, 2001; Nivre, 2008)	Yes	SH	0
Easy-first (Goldberg and Elhadad, 2010)	No		
Attardi (Attardi, 2006)	Yes	SH	0
Planar (Gómez-Rodríguez and Nivre, 2010)	Yes	SH	1
2-Planar (Gómez-Rodríguez and Nivre, 2010)	Yes	SH	1
Arc-eager with buffer transitions (Fernández-G and Gómez-R, 2012)	Yes	SH	2,3
Swap (Nivre, 2009)	No		
Swap-hybrid (Lhoneux, Stymne, and Nivre, 2017a)	No		
Arc-swift (Qi and Manning, 2017)	Yes	SH, RA_k	1
Spinal arc-eager (Ballesteros and Carreras, 2015)	Yes	SH, RA	1
(Yamada and Matsumoto, 2003)	No		
(Choi and Palmer, 2011)	Yes	SH	1
(Choi and McCallum, 2013)	Yes	No-SH, R-SH	1
Non-monotonic arc-eager (Honnibal, Goldberg, and Johnson, 2013)	Yes	SH, RA	1
Improved non-monotonic arc-eager (Honnibal and Johnson, 2015)	No		
Non-monotonic Covington (Fernández-G and Gómez-R, 2017)	Yes	SH	1
Tree-constrained arc-eager (Nivre and Fernández-González, 2014)	No		
Non-local Covington (Fernández-G and Gómez-R, 2018)	Yes	SH	1
Two-register (Pitler and McDonald, 2015)	No		
Stack-pointer (Ma et al., 2018)	No		
Left-to-right pointer network (Fernández-G and Gómez-R, 2019)	No		

Table 25: An overview of some of the existing transition-based parsers based on the mapping criteria for sequence labeling parsing, such as: being a left-to-right system (L2R?), containing read transitions, and the value of the constant k. The latter serves rather as a guide, as k can vary between parser's variants.

5.4 Models and experiments

In this section, we will test empirically whether sequence labeling parsers that rely on encodings derived from transition-based systems provide a competitive performance in comparison with the head selection and bracketing-based family. Firstly, we will analyze the facets of the transition-based encodings and secondly, we will investigate the overall accuracy of the encodings tested under various setups.

Model We will rely on two different encoders, i.e. BiLSTM and BERT as in the setup in Section 4.4.1.2. Moreover, following Section 3.3.2, we apply MTL, where the models output for each input token two labels: one corresponding to the most probable subsequence of transitions (Task 1) and second one with the dependency relation type (Task 2). As in Section 4.4.1.1, we also conduct experiments with two setups: (i) PoS tag vectors are used in the input for the BiLSTM-based models (denoted as PoS+) and (ii) without PoS tag embeddings (PoS–). We

remind that BERT-based models do not rely on PoS tags by default. The architectures are illustrated in Figure 21 and the hyperparameters for the BiLSTM-based and BERT models are specified in Section A.1.1 and Section A.1.3, respectively.

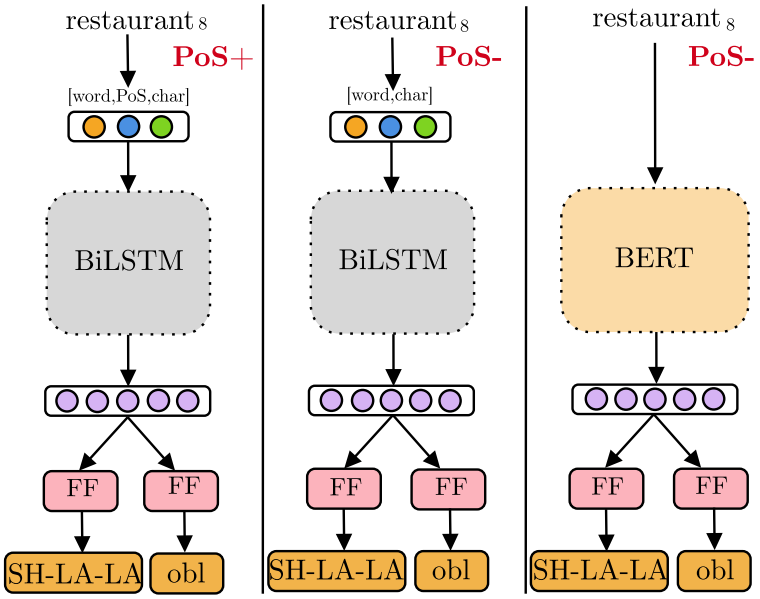


Figure 21: Architectures for learning transition-based labels as two tasks in the MTL setup relying on the BiLSTM-based and BERT architectures with (PoS+) and without PoS tag embeddings (PoS-).

Setup As described earlier, we will test our mapping method using four left-to-right transition-based systems: arc-standard, arc-eager, arc-hybrid and the non-projective Covington. Following Anderson and Gómez-Rodríguez (2020a), we test the encodings on a representative subset of UDv.2.4 languages. More specifically, we select: Ancient $\text{Greek}_{\text{Perseus}}$, $\text{Chinese}_{\text{GSD}}$, $\text{English}_{\text{EWT}}$, $\text{Finnish}_{\text{TDT}}$, $\text{Hebrew}_{\text{HTB}}$, $\text{Russian}_{\text{GSD}}$, $\text{Tamil}_{\text{TTB}}$, $\text{Uyghur}_{\text{UDT}}$ and $\text{Wolof}_{\text{WTB}}$ and using UDPipe (Straka and Straková, 2017) we obtain the predicted segmentation, tokenization and PoS tags.

5.4.1 Encodings' facets

Now, we will perform an analysis of various aspects of the transition-based encodings. Firstly, we will examine whether their differing definitions of transition operators trigger divergence in the label distribution. In addition, we will compare the sizes of label sets against those coming from the other families introduced in previous chapters. Secondly, we will show that sequence labeling parsers obviate the need of using

stack-based *positional features* that provide information about the position of the elements on the stack, and which are crucial for predicting transitions in transition-based parsers.

Treebank	Encoding	Task 1	Task 2
Ancient Greek$_{\text{Perseus}}$	rel-PoS	166	27
	1p-brackets	210	27
	arc-standard	46	27
	arc-eager	501	27
	arc-hybrid	41	27
	Covington	3651	27
Chinese$_{\text{GSD}}$	rel-PoS	201	45
	1p-brackets	104	45
	arc-standard	56	45
	arc-eager	817	45
	arc-hybrid	54	45
	Covington	4846	45
English$_{\text{EWT}}$	rel-pos	202	51
	1p-brackets	109	51
	arc-standard	90	51
	arc-eager	509	51
	arc-hybrid	52	51
	Covington	2804	51
Finnish$_{\text{TDT}}$	rel-PoS	220	47
	1p-brackets	107	47
	arc-standard	77	47
	arc-eager	356	47
	arc-hybrid	46	47
	Covington	2185	47
Hebrew$_{\text{HTB}}$	rel-pos	162	40
	1p-brackets	107	40
	arc-standard	76	40
	arc-eager	469	40
	arc-hybrid	58	40
	Covington	2070	40
Russian$_{\text{GSD}}$	rel-PoS	160	44
	1p-brackets	93	44
	arc-standard	65	44
	arc-eager	385	44
	arc-hybrid	45	44
	Covington	1573	44
Tamil$_{\text{TTB}}$	rel-PoS	68	31
	1p-brackets	49	31
	arc-standard	22	31
	arc-eager	79	31
	arc-hybrid	25	31
	Covington	543	31
Uyghur$_{\text{UDT}}$	rel-PoS	102	44
	1p-brackets	58	44
	arc-standard	20	44
	arc-eager	185	44
	arc-hybrid	26	44
	Covington	1459	44
Wolof$_{\text{WTB}}$	rel-PoS	102	40
	1p-brackets	69	40
	arc-standard	69	40
	arc-eager	373	40
	arc-hybrid	46	40
	Covington	920	40

Table 26: Comparison of sizes of label sets generated by encodings from the different families based on the training and dev set of the UD treebanks. Task 1 corresponds to the labels that encode the arcs and Task 2 to the dependency relation type.

5.4.1.1 Label set size

We begin with investigating the number of labels generated by each transition-based encoding, including those coming from different encoding families. Table 26 reports the output label set for each subtask based on the training and dev set across a subset of UD treebanks. As observed earlier (see Section 4.3.1.2), the relaxed 1-planar bracketing-based encoding tends to produce less labels than the relative PoS-based one. However, when expanding the comparison to the transition-based encodings, it is easily discernible that the ARC-STANDARD and ARC-HYBRID models operate on the smallest output vocabulary of all encodings. Interestingly, the ARC-EAGER encoding has a considerably larger labels space than the counterparts, except the COVINGTON encoding that results in a greatly outsized output vocabulary which may cause sparsity issues.

5.4.1.2 Positional features

Transition-based systems can be compared in terms of the number of positional features they require in order to learn a transition well. For instance, Shi, Huang, and Lee (2017) propose a minimal feature set for BiLSTM-based dependency parsers and show that in the arc-standard model, positional features may be reduced to three: two stack s_0, s_1 features and one buffer b_0 feature in order to achieve competitive results. For the arc-eager and arc-hybrid models, in turn, two positional features suffice: one stack s_0 and one buffer b_0 feature. Unlike transition-based systems that learn a transition at a time, a sequence labeling parser learns labels corresponding to a sequence of transitions and it only needs to access the first buffer word b_0 to assign a label. In other words, our sequence labeling parser does not use any stack-based features in its prediction. In Table 27 we compare the accuracy of both systems, although it only serves for orientation purposes since they differ in hyperparameter selection.

Model	Features	arc-standard	arc-eager	arc-hybrid
Shi et al.	$\{s_2, s_1, s_0, b_0\}$	$93.95_{\pm 0.12}$	$93.92_{\pm 0.04}$	$94.08_{\pm 0.13}$
	$\{s_1, s_0, b_0\}$	$94.13_{\pm 0.06}$	$93.91_{\pm 0.07}$	$94.08_{\pm 0.05}$
	$\{s_0, b_0\}$	$54.47_{\pm 0.36}$	$93.92_{\pm 0.07}$	$94.03_{\pm 0.12}$
	$\{b_0\}$	$47.11_{\pm 0.44}$	$79.15_{\pm 0.06}$	$52.39_{\pm 0.23}$
Our model	$\{b_0\}$	92.13	93.22	92.66

Table 27: Performance of the sequence labeling models with transition-based encodings that require a single positional feature b_0 compared with the parser of Shi, Huang, and Lee (2017). The models are evaluated in UAS (%) on the English PTB dev set.

In general, our sequence labeling parser that obviates the need of the stack-based positional features yields competitive results, when com-

paring against the transition-based system that only relies on the b_0 feature (for instance, the arc-eager models obtain 92.13% and 47.11%, respectively). Although, when the transition-based models make use of the stack-based features, they indeed obtain higher results than the sequence labeling parser.

5.4.2 Performance comparison of the transition-based encodings

We contrast the accuracy of the transition-based encodings against the head selection and bracketing-based encodings in order to obtain an overall comparison across all encoding families.[2] Specifically, we test the encodings in various setups: when PoS tag embeddings are used as input or not, and when applied with different encoders. The reason for that is that in the previous experiments (see Section 4.4.1.1 and Section 4.4.1.2), it has been shown that lack of PoS tags as input features and use of different architectures may have impact on the encoding performance revealing under which setup each encoding type excels most.

5.4.2.1 Impact of using PoS tags as input features

We turn now to the analysis of the results obtained with the BiLSTM-based models in both PoS+ and PoS– setups. Table 28 provides the UAS and LAS scores of each encoding across different treebanks along with their averaged results. In general, the transition-based encodings achieve comparable results to those of the relative PoS-based and the 1-planar bracketing-based encodings. It is most prominent in the PoS– setup, where one of the transition-based encodings (ARC-EAGER) even outperforms the rest of the encodings for some of the treebanks. Nevertheless overall, in the PoS+ setup the relative PoS-based encoding remains on average the best one, while in the PoS– setup the 1-planar bracketing-based encoding obtains the highest accuracy. However, in the latter setup, most of the transition-based models (except for COVINGTON) also outperform on average the relative PoS-based models.

When contrasting solely the transition-based encodings, one may observe that the ARC-STANDARD, ARC-EAGER and ARC-HYBRID models perform similarly with a small advantage of ARC-EAGER. Interestingly, the latter has a considerably larger label set size than the counterparts, as shown in Table 26. COVINGTON, in contrast, underperforms in a range of $1.5 - 2.1\%$ UAS (avg) with respect to the other transition-based models on the test sets, for both settings. This may be explained by the

2 For comparison, we only include the relative PoS-based and the relaxed 1-planar bracketing-based encodings since they provide representative models of each family. It is also worth mentioning that this work was done in parallel with the work on the 2-planar variants.

Split	Encoding	Ancient Greek		Chinese		English		Finnish		Hebrew		Russian		Tamil		Uyghur		Wolof		Avg	
		UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
dev_{PoS+}	rel-PoS	**70.2**	**61.8**	61.6	57.4	**82.6**	**79.0**	**83.2**	**79.3**	**67.4**	**63.8**	**84.0**	**79.6**	**59.5**	**52.0**	72.6	59.0	**77.0**	**70.5**	***73.1***	***66.9***
	1p-brackets	65.2	57.2	**61.8**	**57.9**	82.2	78.6	82.8	79.1	67.1	63.4	83.5	78.9	57.4	50.2	**73.1**	**59.9**	76.3	69.7	*72.2*	*66.1*
	arc-standard	62.2	54.7	60.3	56.2	79.9	76.3	79.7	76.0	64.2	60.5	80.0	75.8	57.8	49.7	72.5	59.2	73.0	66.8	*69.9*	*63.9*
	arc-eager	60.4	52.9	60.9	56.9	80.8	77.3	80.2	76.4	65.4	61.8	81.5	77.4	57.9	50.2	73.2	59.9	74.6	68.0	*70.5*	*64.5*
	arc-hybrid	62.7	55.2	60.3	56.3	80.5	76.9	80.4	76.7	65.2	61.6	81.1	76.4	58.1	50.5	72.9	59.5	72.9	66.5	*70.5*	*64.4*
	Covington	64.0	56.0	56.1	52.3	79.6	75.9	80.7	76.6	64.7	61.1	80.0	75.7	53.2	45.7	67.2	55.0	71.7	65.3	*68.6*	*62.6*
$test_{PoS+}$	rel-PoS	**69.5**	**60.0**	**64.4**	**60.0**	**82.0**	**78.5**	80.2	74.9	**63.0**	**59.5**	82.7	77.3	**58.0**	**49.8**	71.5	57.8	**76.1**	**69.6**	***71.9***	***65.3***
	1p-brackets	64.9	56.2	64.2	59.8	81.7	78.3	**81.1**	**76.0**	62.1	58.7	**82.9**	**77.9**	57.2	49.0	**72.0**	**58.5**	75.4	68.8	*71.3*	*64.8*
	arc-standard	61.0	52.6	62.9	58.5	79.2	75.8	78.5	73.4	60.3	56.5	79.1	74.2	57.1	48.6	71.5	58.2	72.0	65.7	*69.1*	*62.6*
	arc-eager	60.4	52.0	63.2	59.0	80.2	76.7	79.0	73.6	61.2	57.6	80.4	75.3	56.2	48.3	71.9	58.7	73.5	67.1	*69.6*	*63.1*
	arc-hybrid	62.1	53.5	63.2	58.2	79.5	76.1	78.9	74.0	60.4	56.8	79.3	74.4	57.0	48.7	71.3	58.6	72.2	65.7	*69.3*	*62.9*
	Covington	63.7	54.4	58.3	54.2	78.5	75.0	78.9	73.7	60.2	56.6	79.2	74.0	52.3	44.8	66.4	53.5	69.9	63.8	*67.5*	*61.1*
dev_{PoS-}	rel-PoS	**65.3**	**58.3**	58.8	55.3	80.3	77.1	80.8	77.3	65.3	62.2	81.3	77.4	52.7	41.9	65.7	53.3	73.1	66.3	*69.2*	*63.2*
	1p-brackets	64.7	57.2	**63.8**	**59.4**	**83.4**	**80.0**	**84.1**	**80.4**	**68.7**	**65.0**	**84.0**	**79.7**	55.9	45.1	72.1	**58.8**	**75.2**	**67.4**	***72.4***	***65.9***
	arc-standard	61.6	54.8	61.9	57.6	80.9	77.5	80.7	77.2	65.9	62.6	80.3	76.2	55.0	43.3	71.6	58.2	70.4	63.2	*69.8*	*63.4*
	arc-eager	60.0	53.2	63.4	59.2	81.9	78.5	81.6	78.2	66.9	63.2	81.7	77.4	**56.4**	**45.5**	**72.3**	**58.8**	73.4	66.1	*70.9*	*64.4*
	arc-hybrid	62.3	55.5	62.4	58.1	81.4	78.0	81.3	77.9	66.1	62.6	81.0	76.9	56.3	44.9	71.7	58.1	71.9	64.8	*70.5*	*64.1*
	Covington	63.7	56.1	57.4	53.5	80.5	76.8	81.8	78.0	66.3	63.0	80.0	75.7	51.1	40.8	67.0	54.1	71.3	63.6	*68.8*	*62.4*
$test_{PoS-}$	rel-PoS	62.9	**55.1**	60.3	56.8	78.5	75.3	65.6	59.6	59.7	56.6	79.0	73.8	51.6	40.6	64.9	52.4	72.3	65.6	*66.1*	*59.5*
	1p-brackets	**63.4**	54.8	**65.3**	**61.1**	**82.4**	**78.9**	**75.1**	**67.7**	**62.3**	**58.6**	**81.8**	**75.8**	**56.9**	**42.6**	71.0	57.4	**75.1**	**67.2**	***70.4***	***62.7***
	arc-standard	59.5	52.0	64.2	59.9	79.3	75.8	73.4	66.4	60.1	56.4	78.6	73.1	53.8	41.3	70.5	56.9	71.1	63.6	*67.8*	*60.6*
	arc-eager	59.5	51.7	64.5	60.5	80.0	76.6	72.3	64.1	61.1	57.3	79.7	74.0	53.4	41.1	**71.6**	**57.9**	72.9	65.2	*68.3*	*60.9*
	arc-hybrid	60.4	52.5	64.2	59.9	80.1	76.7	74.7	67.5	60.6	57.0	78.6	73.1	53.7	41.7	71.1	57.6	71.5	63.7	*68.3*	*61.1*
	Covington	62.5	53.5	59.6	55.7	78.6	74.9	73.8	66.6	60.6	57.0	78.2	72.5	48.2	36.6	66.3	53.6	69.2	61.6	*66.3*	*59.1*

Table 28: Results for the transition-based encodings with the BiLSTM-based models, in which PoS tags are used as input (PoS+) or are omitted (PoS–).

outsized label space. In theory, the former algorithms run in $\mathcal{O}(n)$ transitions per sentence, while COVINGTON has $\mathcal{O}(n^2)$ transitions that in our approach are grouped into n labels. In practice, the encoding generates a considerably larger label set size that may be harder to learn by the model. On the other hand, COVINGTON outperforms the counterpart transition-based models on the highly non-projective treebank for Ancient Greek, suggesting that it is especially suitable when dealing with non-projectivity, unlike the fully projective transition-based encodings. Although, it still performs worse than the relaxed 1-planar bracketing-based encoding that only partly handles non-projectivity.

5.4.2.2 Impact of using a different architecture

Furthermore, we extend the analysis of the encodings using BERT (without relying on PoS tag vectors as the input features). The results are shown in Table 29. Again, the tendencies observed in the PoS– setup with a BiLSTM-based model remain (see Section 4.4.1.1), where the 1-planar bracketing-based encoding achieves the highest scores averaged across the languages and outperforms the other encodings on almost all treebanks. The ARC-STANDARD, ARC-EAGER and ARC-HYBRID models perform on par and lie behind the bracketing-based one on average by 1.8% and 1.9% UAS on the test set. Furthermore, the non-projective COVINGTON and rel-PoS models have noticeably inferior performance compared to the counterpart encodings. At the ar-

chitectural level, all encodings achieve higher scores with the monolingual pre-trained models (Chinese, English and Finnish) than with the BiLSTM-based encoder in the PoS– setup, while the BiLSTM-based models obtain overall better performance than M-BERT on the remaining treebanks. When looking at the improvements in the accuracy of BERT with respect to the BiLSTM-based models in the PoS– setup (with averaged scores of only languages used in the BERT setup), the ARC-STANDARD, ARC-EAGER and ARC-HYBRID models obtain improvements of 2.9%, 2.6% and 2.4% UAS, respectively. On the other hand, the non-projective COVINGTON models improve merely by 0.2 indicating that the latter is harder to learn also with a different architecture, presumably due to the larger label set size, as discussed previously.

Split	Encoding	$\text{Chinese}_{\text{GSD}}$		$\text{English}_{\text{EWT}}$		$\text{Finnish}_{\text{TDT}}$		$\text{Hebrew}_{\text{HTB}}$		$\text{Russian}_{\text{GSD}}$		$\text{Tamil}_{\text{TTB}}$		Avg	
		UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
dev	rel-PoS	59.8	57.0	82.3	79.9	82.6	79.7	61.8	58.0	77.6	73.2	43.4	33.6	*67.9*	*63.5*
	1p-brackets	**69.3**	**65.3**	**87.2**	**84.4**	**88.6**	**85.1**	**66.6**	**61.7**	**80.6**	**75.4**	**49.8**	**36.8**	***73.7***	***68.1***
	arc-standard	67.9	64.1	85.4	82.6	85.7	82.5	64.4	59.8	77.8	72.7	48.2	35.1	*71.6*	*66.1*
	arc-eager	67.0	63.2	85.3	82.4	86.8	83.5	64.7	59.6	77.8	72.9	49.7	35.2	*71.9*	*66.1*
	arc-hybrid	67.0	63.3	85.5	82.8	86.6	83.1	65.2	60.1	78.1	73.4	47.3	34.0	*71.6*	*66.1*
	Covington	59.7	56.0	81.4	78.8	82.4	79.2	61.9	57.3	73.9	68.9	44.6	31.6	*67.3*	*62.0*
test	rel-PoS	61.5	58.8	81.0	78.8	82.6	80.0	56.9	53.1	77.9	73.2	45.2	34.1	*67.5*	*63.0*
	1p-brackets	**70.2**	**66.5**	**86.2**	**83.3**	**89.4**	**86.1**	**62.7**	**57.9**	**79.9**	**74.2**	48.9	35.9	***72.9***	***67.3***
	arc-standard	68.5	64.7	84.0	81.2	86.1	83.1	59.6	54.3	77.6	72.3	**51.0**	**37.3**	*71.1*	*65.5*
	arc-eager	69.2	65.6	83.8	81.2	87.2	83.9	60.2	55.3	77.6	72.4	48.6	36.7	*71.1*	*65.8*
	arc-hybrid	69.1	65.5	84.1	81.3	86.6	83.6	60.3	55.7	77.5	72.3	48.2	35.6	*71.0*	*65.7*
	Covington	60.1	56.7	79.7	77.4	83.2	80.2	57.6	52.6	73.1	68.0	46.3	34.0	*66.7*	*61.5*

Table 29: Results for the transition-based encodings using BERT as encoder.

5.4.2.3 Encoding evaluation with gold data

In the previous experiments, we evaluated the encodings in real-world settings, where not only PoS tags but also prior steps like segmentation and tokenization were predicted. However, the accuracy of such predicted setup may vary among languages. Table 30 reports the prediction accuracy of UDPipe used in our experiments. Similarly as in Section 4.4.1.3, we examine the impact of the predicted setup on all encodings and provide in Table 31 the upper bounds for models on the gold data sets. We use PoS tags as the input features to the model.

Taken together, the results show that the predicted setup has a relatively large impact on the models. The largest averaged disparity between the gold and predicted data set can be observed in Hebrew (40.2% UAS), Chinese (31.5%) and Tamil (29.4%). In the gold setup, the best performing model in terms of UAS across the selected treebanks is based on the relative PoS-based encoding (in 7 out of 9 treebanks). However, the results also suggest that the same encoding has the largest percentage decrease in the predicted setup. This can be caused due to

Language	Words (%)	Sentences (%)	UPoS (%)
Ancient Greek	100.0	98.9	82.2
Chinese	90.0	99.1	84.0
English	99.0	76.3	93.5
Finnish	99.7	88.6	94.3
Hebrew	85.0	99.4	80.5
Russian	99.5	96.2	95.0
Tamil	94.5	97.5	81.3
Uyghur	99.7	82.9	87.9
Wolof	99.2	92.0	91.7

Table 30: Prediction accuracy of UDPipe 1 on UDv2.4 treebanks.

	Ancient Greek		Chinese		English	
Encoding	UAS	LAS	UAS	LAS	UAS	LAS
rel-PoS	74.8 *(+7.6%)*	68.3 *(+13.9%)*	84.9 *(+31.8%)*	82.0 *(+36.7%)*	89.3 *(+8.9%)*	86.9 *(+10.7%)*
1p-brackets	69.1 *(+6.4%)*	63.1 *(+12.3%)*	84.2 *(+31.1%)*	81.2 *(+35.8%)*	88.8 *(+8.7%)*	86.5 *(+10.4%)*
arc-standard	65.1 *(+6.7%)*	59.6 *(+13.4%)*	82.8 *(+31.6%)*	79.9 *(+36.6%)*	85.7 *(+8.2%)*	83.4 *(+10.0%)*
arc-eager	64.3 *(+6.4%)*	58.6 *(+12.6%)*	83.7 *(+32.4%)*	81.1 *(+37.4%)*	86.6 *(+8.0%)*	84.2 *(+9.8%)*
arc-hybrid	66.2 *(+6.6%)*	60.3 *(+12.7%)*	83.0 *(+31.4%)*	80.1 *(+36.2%)*	86.5 *(+8.8%)*	84.2 *(+10.7%)*
Covington	67.9 *(+6.6%)*	61.6 *(+13.2%)*	76.3 *(+30.8%)*	73.4 *(+35.4%)*	85.1 *(+8.5%)*	82.8 *(+10.4%)*
avg	*(+6.7%)*	*(+13.0%)*	***(+31.5%)***	***(+36.4%)***	*(+8.5%)*	*(+10.3%)*
	Finnish		Hebrew		Russian	
Encoding	UAS	LAS	UAS	LAS	UAS	LAS
rel-PoS	85.1 *(+6.2%)*	81.3 *(+8.5%)*	88.2 *(+40.1%)*	85.4 *(+43.7%)*	86.3 *(+4.3%)*	81.5 *(+5.3%)*
1p-brackets	86.1 *(+6.2%)*	82.5 *(+8.5%)*	87.5 *(+40.9%)*	84.7 *(+44.3%)*	86.0 *(+3.7%)*	81.6 *(+4.8%)*
arc-standard	83.1 *(+5.9%)*	79.6 *(+8.5%)*	84.2 *(+39.7%)*	81.4 *(+43.9%)*	82.2 *(+3.9%)*	77.9 *(+4.9%)*
arc-eager	83.9 *(+6.2%)*	80.0 *(+8.6%)*	85.6 *(+39.8%)*	82.6 *(+43.6%)*	83.7 *(+4.1%)*	79.2 *(+5.2%)*
arc-hybrid	83.5 *(+5.8%)*	80.1 *(+8.2%)*	85.1 *(+40.9%)*	82.2 *(+44.8%)*	82.4 *(+4.0%)*	78.1 *(+5.0%)*
Covington	83.5 *(+5.8%)*	79.7 *(+8.1%)*	84.0 *(+39.5%)*	81.1 *(+43.3%)*	82.2 *(+3.8%)*	77.8 *(+5.1%)*
avg	*(+6.0%)*	*(+8.4%)*	***(+40.2%)***	***(+43.9%)***	*(+4.0%)*	*(+5.0%)*
	Tamil		Uyghur		Wolof	
Encoding	UAS	LAS	UAS	LAS	UAS	LAS
rel-PoS	76.5 *(+31.9%)*	68.7 *(+37.9%)*	74.3 *(+4.0%)*	61.1 *(+5.7%)*	86.0 *(+13.0%)*	81.8 *(+17.5%)*
1p-brackets	73.6 *(+28.5%)*	66.8 *(+36.3%)*	74.7 *(+3.7%)*	61.7 *(+5.5%)*	85.3 *(+13.1%)*	80.7 *(+17.2%)*
arc-standard	73.3 *(+28.2%)*	66.7 *(+37.2%)*	74.4 *(+4.0%)*	61.7 *(+6.0%)*	80.6 *(+11.9%)*	76.5 *(+16.6%)*
arc-eager	74.6 *(+32.8%)*	67.6 *(+40.1%)*	74.3 *(+3.3%)*	61.3 *(+4.3%)*	82.5 *(+12.2%)*	78.1 *(+16.4%)*
arc-hybrid	72.8 *(+27.7%)*	66.0 *(+35.5%)*	73.6 *(+3.2%)*	61.5 *(+5.0%)*	80.8 *(+11.9%)*	76.5 *(+16.5%)*
Covington	66.6 *(+27.4%)*	59.9 *(+33.8%)*	68.7 *(+3.5%)*	56.5 *(+5.7%)*	78.1 *(+11.7%)*	74.2 *(+16.3%)*
avg	***(+29.4%)***	***(+36.8%)***	*(+3.6%)*	*(+5.4%)*	*(+12.3%)*	*(+16.7%)*

Table 31: Upper bounds for encodings using the gold segmentation, tokenization and PoS tags on the test sets of UD treebanks. In parenthesis the percentage of increase with respect to the predicted setup is given.

the strong reliance on PoS tags. Nevertheless, it seems that the prediction accuracy of the upstream tasks in general considerably decreases the score of the models but the impact is similar across the encodings.

5.5 Strengths and limitations

One of the key advantages of the transition-based encodings is that they establish a theoretical link between the transition-based and sequence labeling parsing. We have shown that our approach can be applied to a wide range of left-to-right transition-based algorithms, also beyond the ones used in the experiments. It implies that new encodings can be easily retrieved from the existing algorithms and used in fast and simple sequence labeling parsing. An additional strength of this

approach, although falling outside the scope of this thesis, is that it can be further extended to transition-based constituent and semantic parsers.

Furthermore, the theoretical link between these two approaches is strengthened with the empirical results showing that the transition-based encodings are learnable and yield comparable performance with the ones coming from different encoding families. Unlike the relative PoS-based encoding, they do not rely on any additional features. Moreover, some of them (ARC-STANDARD and ARC-HYBRID) tend to generate the smallest output vocabulary, ensuring a compact label vocabulary.

However, we observe that transition-based systems that do not run in $\mathcal{O}(n)$ transitions per sentence may considerably expand the output vocabulary size. This, in turn, may impede effective learning and result in models with a lower accuracy, as in the case of the COVINGTON algorithm, even though it has an advantage of supporting non-projectivity.

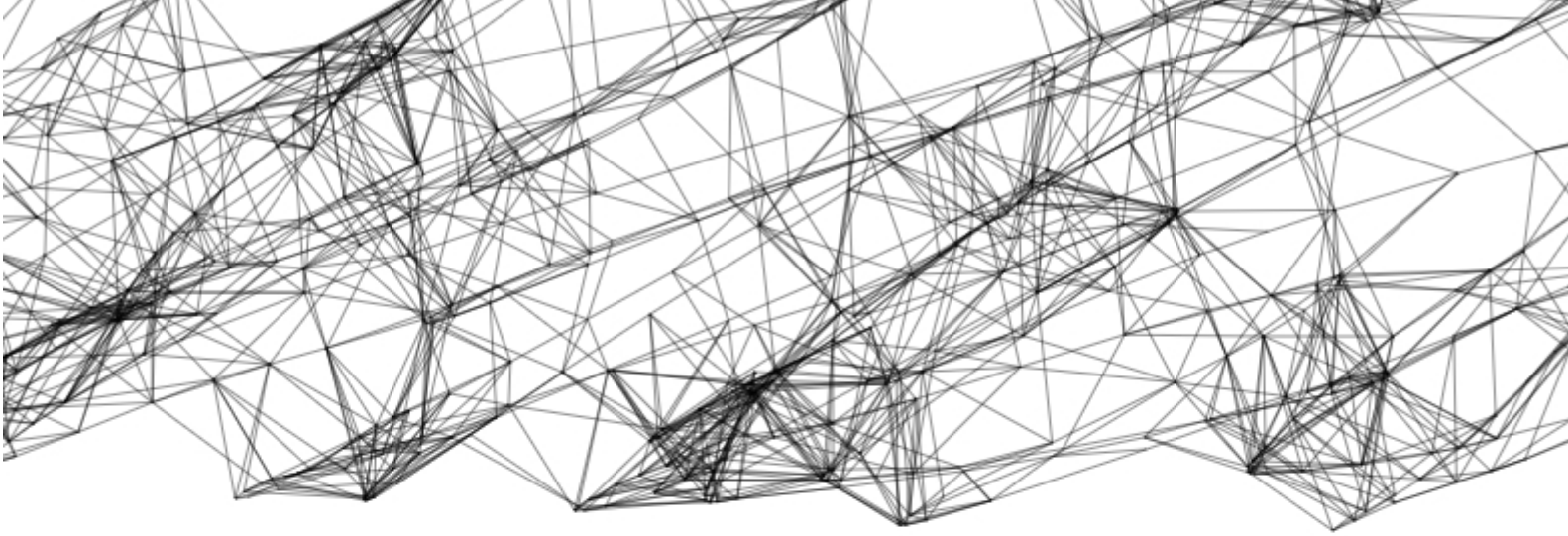
5.6 Conclusions

In this chapter, we have introduced a set of new encodings obtained automatically from some existing transition-based algorithms. First, we have shown how transition-based parsers can be cast into sequence labeling parsers. More specifically, we have established a method to split the transition sequence based on read transitions in order to map it into a sequence of labels of the same length as the input sentence. We have also shown that this method is applicable to a wide range of existing left-to-right transition-based algorithms.

In the second part of this chapter, we have tested the transition-based encodings empirically. Firstly, we have investigated their facets. Regarding the output vocabulary size, two encodings (ARC-STANDARD and ARC-HYBRID) have shown to provide very compact representations of dependency trees. Contrarily, the non-projective COVINGTON seems to suffer more from the excessive label set size that has a negative impact on the UAS and LAS scores across various treebanks. Interestingly, the experiments have shown that the differences across the transition-based systems, such as transition operators and how they manipulate stack and buffer, are also prominent in sequence labeling parsing expressed in terms of label set size and the ease of learning them. Moreover, we have shown that our approach, unlike the transition-based parsers, does not require any stack-based features in order to predict the sequence of transitions.

Furthermore, we have examined the accuracy of the encodings in two testing settings, where we have used or omitted PoS tag embeddings as input features. As mentioned before, the reason for defining such setups is that in certain cases using PoS tags is not preferable. For instance, they may be not available in the low-resource languages or have low prediction accuracy. One may also not want to include a PoS tagger

in the pipeline in order to reduce the computational cost or simply, the selected architecture does not make use of this type of embeddings, as in the case of BERT. Regarding the latter, we have included it in our experiments to investigate the impact of using a different architecture on encodings. In general, the results have confirmed that the transition-based models (except COVINGTON) obtain comparable performance with the models from the head selection and bracketing-based families.



Part III

LEVERAGING COMPLEMENTARY DATA

Further performance enhancements of sequence labeling parsers can be obtained by leveraging representations from external complementary data. By means of multi-task learning (MTL), the models are augmented with representations coming from (i) constituency trees and (ii) eye-tracking measures.

6 Joint learning of dependency and constituency representations

In this chapter, we will explore whether complementary data may enrich representations learned by our sequence labeling parsers. More specifically, the aim is to further improve the performance of the sequence labeling parsers by joint learning of dependency and constituency paradigms using MTL, while preserving high parsing efficiency. To do so, we will take advantage of the existing encoding methods for constituency parsing as sequence labeling, proposed by Gómez-Rodríguez and Vilares (2018), that can be readily integrated with our model.

We will first recall how constituency structures can be represented as linearized trees. Afterwards, we will propose models in various setups to leverage the shared representations of both paradigms. Finally, we will provide empirical results of the models and compare them against existing parsers.

6.1 Dependency and constituency parsing

We begin with a brief outline of the complementary nature of the two paradigms and the attempts that have been undertaken at leveraging both of them in syntactic parsing. Thereafter, we will refer to the encoding methods in dependency and constituency parsing as sequence labeling.

6.1.1 Paradigms' complementarity

A syntactic structure of a sentence can be described in accordance with the dependency paradigm (Mel'cuk, 1988), such it has been used in this thesis, but there are also other paradigms to represent the syntactic properties of sentences, for instance in the form of constituents (Chomsky, 1956). Each representation exhibits some peculiarities and the information they convey about the syntactic structure of an arbitrary sentence is not equivalent (Kahane and Mazziotta, 2015). In general, the syntactic analyses with respect to the two paradigms differ in how they represent the syntactic interplay of the words in a sentence. As explained in Section 2.1, the dependency grammar captures binary asymmetric relations between word pairs (in terms of a head and a dependent that are linked with a dependency relation), while in the constituency grammar, in contrast, words are grouped according

to the phrase hierarchy. Examples of a dependency and constituency tree are shown in Figure 22 and Figure 23, respectively. When it comes to the complementary nature of the two paradigms, a dependency tree, for instance, does not implicitly denote the word's correspondence to phrase chunks, while a constituency tree does not contain information about the dependency relation type between word pairs.

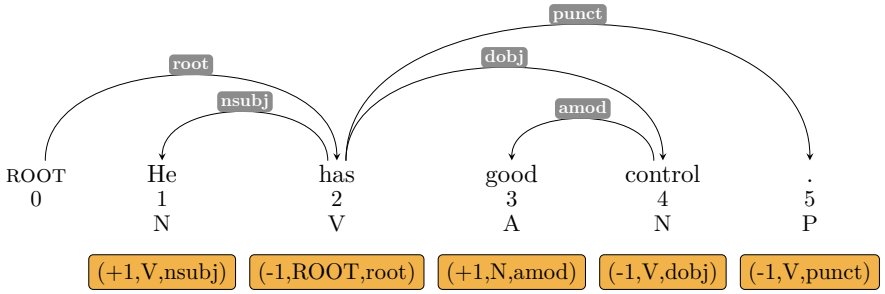


Figure 22: An encoded dependency tree.

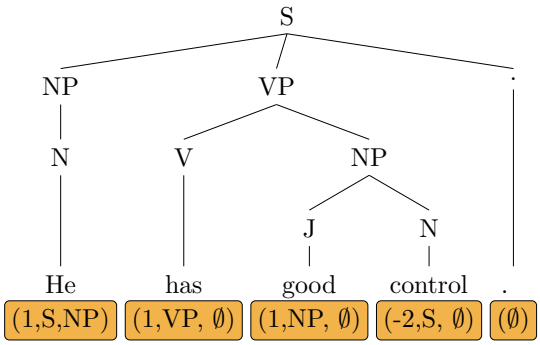


Figure 23: An encoded constituency tree.

In the context of syntactic parsing, the advances of parsers relying on each formalism are usually kept separately. Hence, many SOTA systems fall into the group of either dependency or constituency parsing. However, a few attempts have been made to make parsers benefit from the complementary representations of the two abstractions to improve their accuracy. For instance, in one of the pre-deep learning approaches, Ren, Chen, and Kit (2013) combine dependency and constituency parsing in order to choose the most plausible parse. To do so, first a probabilistic context free grammar is applied to obtain the n best constituency trees and then a dependency parser is used to score and re-rank both types of trees. Alternatively, Klein and Manning (2002) introduce a factored model that obtains the most optimal parses by first scoring phrase-structure and lexical dependency trees with separate models and then combining them. However, these approaches were primarily intended for statistical parsers. Most recently (as at the time of writing this thesis), the idea of combining the two paradigms revived and ensemble models based on deep neural architectures have shown

to provide considerable improvements. For instance, Zhou and Zhao (2019) obtain SOTA results on both tasks by including the dependency and constituency representations in the head-driven phrase structure (HPSG) augmented with representations from BERT and XLNet (Yang et al., 2019). In the same vein, Mrini et al. (2020) demonstrate that joint learning of both paradigms by a neural model leads to further enhancements in accuracy. Similarly, Fernández-González and Gómez-Rodríguez (2020) show that competitive results can be obtained by training a single model with dependency and constituency representation in the MTL setup using the Pointer Network architecture (Vinyals, Fortunato, and Jaitly, 2015).

6.1.2 Encodings

Now, we will describe the linearization methods of syntactic trees used in sequence labeling parsing. For completeness, we will formulate them for both paradigms. It is also worth emphasizing that we will make use of the relative PoS-based encoding in the experiments, however any of the proposed encodings can be applied in this framework.

Dependency tree encoding As introduced in Chapter 3, a dependency tree can be represented as sequence of labels for an arbitrary sentence of length $|w|$ by applying a linearization method: $\Pi_{|w|} : T_{d,|w|} \rightarrow L_d^{|w|}$. Following the definitions from Section 3.1.3, each label in the relative PoS-based encoding consists of three components (o_i, p_i, d_i) that encode for a given word w_i the relative distance o_i to the head's PoS tag p_i with a dependency relation d_i. An example of such label assignment is shown in Figure 22.

Constituency tree encoding Following Gómez-Rodríguez and Vilares (2018), we define a linearization method for constituency trees as: $\Phi_{|w|} : T_{c,|w|} \rightarrow L_c^{|w|}$. Similarly, each label corresponds to an input token w_i and is composed of three elements (n_i, c_i, u_i). Let a_i be the number of non-terminal ancestors that are shared between the focus word w_i and the following one w_{i+1}. Then, the first component n_i is computed by taking the difference between a_i and a_{i-1}, where for a_0, $a_0 = n_0$. For instance, as shown in Figure 23, the value of n_i in the label for the token $control_4$ is -2, since the previous word $good_3$ shares three non-terminal ancestors with $control_4$, while $control_4$ shares one ancestor with the next token. The difference between these two values results in -2. The next element c_i encodes the lowest level non-terminal symbol that is shared between w_i and w_{i+1}, while u_i stands for the leaf unary chain from c_i to w_i (if present).

6.2 Models and experiments

Following the successful application of MTL to a wide range of NLP tasks (Goot et al., 2021), our study aims at examining whether dependency and constituency sequence labeling parsers may benefit from joint learning. The purpose of using the complementary representations of constituency trees in the context of dependency parsing (and vice versa) can be defined twofold. First, learning constituency labels can be treated as a *weighted auxiliary task* (see Section 2.6), where the goal is to let the model access the constituency representations and thereby improve the prediction of dependency labels defined as the *main task*. In this case, the weights determine the prevalence of constituency representations in the model and are adjusted empirically. When treating constituency label prediction as an auxiliary task, we do not compute its output labels, since they are not of interest. Additionally, it enables reducing the prediction time by omitting to execute the corresponding *softmax*. Secondly, the prediction of both dependency and constituency labels may be considered as the main tasks, such that parsing of both paradigms can be performed with a single model.

6.2.1 Models for learning across representations

In the experiments, we will distinguish between single- and double-paradigm models. The former only relies on a single parsing paradigm, while the latter leverages the complementary representations from the second paradigm either as the auxiliary task or the main task. More specifically, we define the following models:

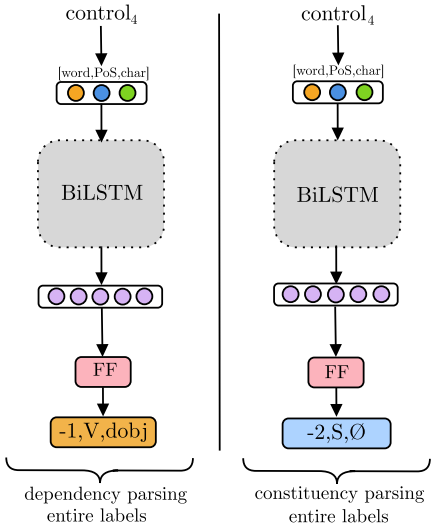


Figure 24: Architecture for the single-paradigm, single-task models (S-S) in dependency and constituency parsing.

Single-paradigm, single-task models (S-S) In this setup, we train two separate models corresponding to each of the parsing paradigms. Every model learns the entire labels as a single task, i.e. a tuple (o_i, p_i, d_i) is learned as a single label for dependency parsing and a tuple (n_i, c_i, u_i) for constituency parsing. An outline of the architecture is shown in Figure 24.

Single-paradigm, multi-task models (S-MTL) For each paradigm we train a model, where every label component corresponds to a separate subtask. In the case of dependency parsing, the partial labels consist of (o_i, p_i) and (d_i), as described in Chapter 3, the relative PoS-based encoding obtains the best performance when learned as two tasks. In constituency parsing, we follow Vilares, Abdou, and Søgaard (2019) and each label is defined as three tasks, i.e. (o_i), (p_i) and (d_i). The loss is computed as $\mathcal{L}{=}\sum_t \mathcal{L}_t$, where $\mathcal{L}_t$ is the partial loss from the subtask t. As discussed in Section 3.3.2, learning labels in the multi-task setup has shown to be beneficial for the model and may serve as a compression method that reduces the output vocabulary and thereby prevents label sparsity. An example of the architecture is illustrated in Figure 25.

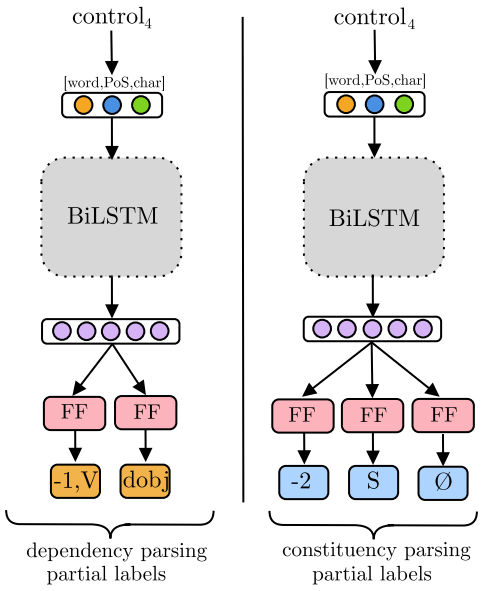


Figure 25: Architecture for the single-paradigm, multi-task models (S-MTL), where labels in dependency parsing are learned in a two-task setup, while in constituency parsing in a three-task setup.

Double-paradigm, multi-task models with auxiliary losses (D-MTL-AUX) In this setup, the predictions of partial labels for one parsing paradigm are defined as the main tasks, while the labels of the counterpart paradigm are considered as auxiliary tasks. The auxiliary tasks only serve for sharing the representations of the complementary

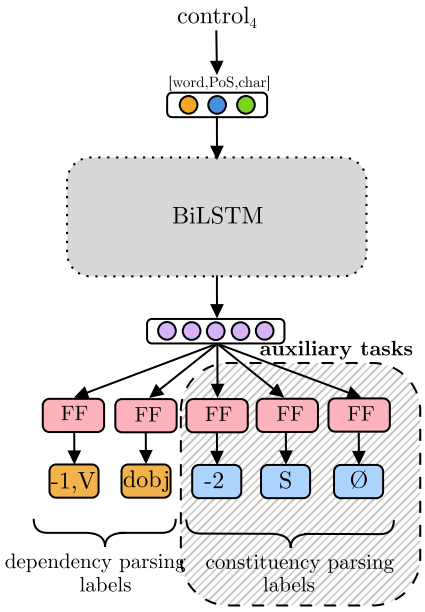


Figure 26: Architecture for the double-paradigm, multi-task models with auxiliary losses (D-MTL-AUX), where the partial dependency labels are learned as the main tasks and constituency labels as auxiliary tasks. Analogously, it can be applied for constituency parsing as the main task.

abstraction and whose output labels are not considered for the end task. During training, the best model is chosen based on the evaluation metrics of the main task i.e. LAS score for dependency parsing and F1 score for constituency parsing. The weighting factor for dependency parsing as an auxiliary task is set to 0.1, while for constituency parsing 0.2. The weighting factors were chosen based on a parameter search in an interval [0,1] with a step of 0.1. In this setup, the loss is computed as $\mathcal{L}{=}\sum_t \mathcal{L}_t + \sum_a \beta_a \mathcal{L}_a$, where $\mathcal{L}_a$ stands for the auxiliary loss and β_a for the weighting factor. We train two such models, where each of them considers one parsing paradigm as the main task and the counterpart paradigm as the auxiliary task. An outline of the architecture used for dependency parsing as the main task is shown in Figure 26.

Double-paradigm, multi-task model (D-MTL) In this model, all labels are learned as the main tasks with an equal weighting factor. This setup enables performing dependency and constituency parsing using a single model. During training the best models are chosen based on the highest harmonic mean among LAS and F1 scores in order to prioritize models that are balanced instead of models that sacrifice accuracy in one task to excel in the other. An example of the architecture is illustrated in Figure 27.

In our approach, the dependency and constituency representations are leveraged with hard parameter sharing, where hidden layers captur-

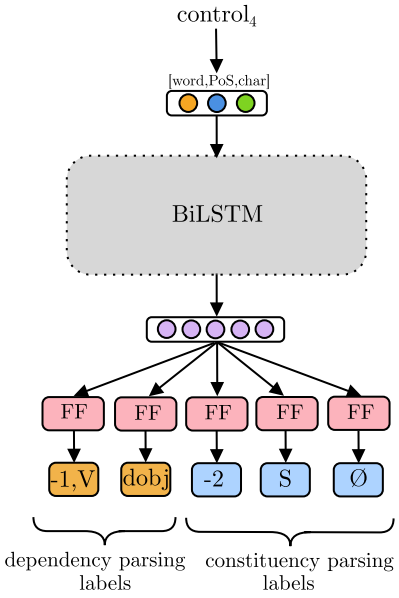


Figure 27: Architecture for the double-paradigm, multi-task model (D-MTL), where both dependency and constituency labels are learned as main tasks.

ing both abstractions are shared across the BiLSTM layers. The output hidden layer representation h_i for a given token w_i are fed to separate feed-forward layers with *softmaxes* corresponding to each subtask. For comparison reasons, we include single and multi-task models that do not contain the representations of the counterpart paradigm. The hyperparameters for all models are specified in Section A.1.2.

6.2.2 Data

We carry out experiments on two treebanks containing parallel data, i.e. each sentence is provided with both dependency and constituency annotations. More specifically, we use the following corpora:

PTB The English Penn Treebank with constituency analyses and data with Stanford Dependency (SD) conversion for dependency parsing (see a more detailed description in Section 2.2.2).

SPMRL To extend our experiments to a wider range of languages, we also train and evaluate our models on the SPMRL treebanks. The data comes from the SPMRL 2013 Shared Task[1] on parsing morphologically rich languages (Seddah et al., 2013). SPMRL data set includes 9 morphologically rich and typologically diverse languages provided with both dependency and constituency annotations. In addition, they are given in gold and predicted format (with predicted segmentation, PoS

1 SPMRL 2013 Shared Task: http://www.spmrl.org/spmrl2013-sharedtask.html

tags and morphological features). This data set is especially suitable for our experiments due to the alignment of dependency and constituency annotations in each treebank and it also enables a direct comparison of our models against dependency and constituency parsers from the Shared Task.

Although, during data preparation we observe some discrepancies between the dependency and constituency versions. For instance, there are few examples where tokens differ (e.g. the token `-LSB-` in dependency treebank corresponds to `[` in the constituency file). Hence, in order to evaluate MTL models, we select a single token for both paradigms. Furthermore, the predicted PoS tags also vary in some cases. Figure 28 shows an example of a token *fiskalak* extracted from the Basque treebank, where the predicted PoS tag for that token in the constituency treebank is `ADJ`, while in the dependency treebank it corresponds to `IZE`. This kind of discrepancy is challenging especially in the case of the relative PoS-based encoding that requires uniform PoS tags for the correct decoding. Additionally, we have also noticed differences in the predicted morphological features that we use as input features to the models.

```
(ADJ##lem=fiskal|AZP=ADJ|KAS=ERG|NUM=S## fiskalak)

19 fiskalak fiskal IZE IZE_ARR KAS=ERG|NUM=S 21 ncsubj _ _
```

Figure 28: An example of discrepancies in the PoS tag prediction (highlighted in bold and red) in the dependency and constituency SPMRL Basque treebank. The first line corresponds to the constituency annotation for the token *fiskalak*, followed by a dependency annotation. For the former the predicted PoS tag is ADJ, while for the latter IZE.

To solve the discrepancies that are problematic for the double-paradigm models, we make use of only PoS tags and input features coming from the main task paradigm in the D-MTL-AUX models, while in the D-MTL setup, we train the models twice, each with the corresponding PoS tags and input features.

6.2.3 Experiments

We carry out several experiments in order to test empirically whether sequence labeling parsers may benefit from sharing representations of counterpart analyses. First, we will compare the accuracy obtained with single- and double-paradigm MTL models and secondly, we will measure the parsing speeds to test if the augmented models still maintain a good trade-off between accuracy and computational efficiency. Lastly, we will provide a comparison of our models against existing dependency and constituency parsers evaluated on the PTB and SPMRL treebanks.

6.2.3.1 Model performance

We evaluate the models with the common metrics for each parsing paradigm. More specifically:

- UAS and LAS for dependency parsing, where punctuation is excluded to provide a homogeneous setup for the PTB and SPMRL data sets.
- F_1 score for constituency parsing. The bracketing F-score is used from the original EVALB (together with COLLINS.prm) and EVAL_SPMRL (together with spmrl.prm) official scripts to evaluate constituency trees.

Language	Model	Dependency Parsing UAS	Dependency Parsing LAS	Constituency Parsing F1	Language	Model	Dependency Parsing UAS	Dependency Parsing LAS	Constituency Parsing F1
English (PTB)	S-S	93.60	91.74	90.82	Hungarian	S-S	88.24	84.54	90.10
	S-MTL	93.84	91.83	90.99		S-MTL	88.69	84.54	90.51
	D-MTL-AUX	**94.05**	**92.01**	**91.04**		D-MTL-AUX	**88.99**	**84.95**	90.44
	D-MTL	93.96	91.90	90.51		D-MTL	88.89	84.89	**90.72**
Basque	S-S	86.20	81.70	89.20	Korean	S-S	86.47	84.12	82.63
	S-MTL	**87.42**	81.71	90.54		S-MTL	86.78	84.39	**82.86**
	D-MTL-AUX	87.19	81.73	**90.84**		D-MTL-AUX	**87.00**	**84.60**	82.76
	D-MTL	87.09	**81.77**	90.50		D-MTL	86.64	84.34	82.40
French	S-S	89.13	85.03	79.71	Polish	S-S	91.17	85.64	92.59
	S-MTL	**89.54**	84.89	**80.40**		S-MTL	91.58	85.04	93.17
	D-MTL-AUX	89.52	84.97	80.39		D-MTL-AUX	91.37	85.20	93.36
	D-MTL	89.45	**85.07**	80.24		D-MTL	**92.00**	**85.92**	**93.52**
German	S-S	91.24	88.79	82.52	Swedish	S-S	86.49	80.60	82.56
	S-MTL	91.54	88.75	**83.08**		S-MTL	87.22	80.61	85.16
	D-MTL-AUX	**91.58**	**88.80**	82.97		D-MTL-AUX	**87.24**	80.34	**85.49**
	D-MTL	91.45	88.67	82.87		D-MTL	87.15	**80.71**	85.38
Hebrew	S-S	82.74	75.08	88.68	*average*	S-S	*88.36*	*84.13*	*86.53*
	S-MTL	83.42	74.91	**91.84**		S-MTL	*88.89*	*84.07*	*87.62*
	D-MTL-AUX	**83.90**	**75.89**	91.78		D-MTL-AUX	***88.98***	***84.28***	***87.67***
	D-MTL	82.60	73.73	91.08		D-MTL	*88.80*	*84.11*	*87.47*

Table 32: Performance comparison of single- and double-paradigm models evaluated on the PTB and SPMRL test sets. The averaged scores across languages are given in italics.

Accuracy Table 32 reports the accuracy of dependency and constituency models in four setups, i.e. (i) single-paradigm, single-task: S-S, (ii) single-paradigm, multi-task: S-MTL, (iii) double-paradigm, multi-task models with auxiliary losses: D-MTL-AUX and (iv) double-paradigm, multi-task: D-MTL. In general, D-MTL-AUX models obtain on average the best performance both in dependency and constituency parsing. More specifically, when comparing against the baseline S-S, the D-MTL-AUX models provide on average improvements in dependency parsing of 0.62% UAS and 0.15% LAS, while for constituency parsing the gains correspond to 1.14% of the F1 score. However, it is also worth noting

that some MTL models for dependency parsing obtain, in fact, a lower LAS score than the S-S models (e.g. S-MTL and D-MTL-AUX models for dependency parsing in French). We hypothesize this could be caused by the fact that the MTL models learn combinations of partial labels that in some cases may result in a choice of wrong component combinations. The flexibility of combining labels arbitrarily by a model is in general more beneficial than imposing learning of entire labels that can be sparse, however it may not hold in some particular cases as in this one.

When analyzing the complementary data leverage among the dependency and constituency models more in detail, one may observe that the dependency models based on the D-MTL-AUX setup clearly outperform the remaining models (in 7 out of 9 treebanks based on the UAS score). In constituency models, in turn, this tendency is not that prevalent. In 4 out of 9 treebanks, the best performance is obtained with S-MTL models, while in 3 out of 9 with D-MTL-AUX models. This suggests that in our setup the dependency parser leverages to a greater extent learning the counterpart paradigm as the auxiliary task. Furthermore, we notice that the D-MTL models are capable of providing both dependency and constituency parses but at the expense of lower accuracy compared to the remaining MTL models. However, overall they still perform in most languages better than the baseline S-S and S-MTL models and thereby exposing the general gains of MTL across dependency and constituency representation.

Model	Dependency parsing	Constituency parsing
S-S	$102_{\pm 6}$	$117_{\pm 6}$
S-MTL	$128_{\pm 11}$	$133_{\pm 1}$
D-MTL-AUX	$128_{\pm 11}$	$133_{\pm 1}$
D-MTL	$124_{\pm 1}$	$124_{\pm 1}$

Table 33: Speed in sentences/second on the PTB test set measured on a single core CPU across 5 runs.

Parsing speed Furthermore, we investigate whether the improved accuracy of the models augmented with complementary representations comes at any additional computational cost. The CPU speeds corresponding to each model setup are reported in Table 33. In general, the results show that the MTL models maintain a good trade-off between speed and accuracy. This stems partly from the fact that in the case of D-MTL-AUX models, the auxiliary tasks are not computed at testing time. When it comes to the D-MTL models, they are able to encapsulate two parsing paradigms in a single model, while still providing a comparable speed.

Overall, the results shows that dependency parsing as sequence labeling enables enhancements with complementary data while maintaining its benefits as high speed and architectural simplicity.

6.2.3.2 Comparison against existing dependency and constituency parsers

Now, we move on to the comparison of the best performing dependency and constituency models with auxiliary tasks (D-MTL-AUX), on the English PTB test set from Table 32, against some existing parsers.

Model	Dependency parsing		Constituency Parsing
	UAS	LAS	F1
Chen and Manning (2014)	91.80	89.60	–
Kiperwasser and Goldberg (2016)	93.90	91.90	–
Dozat and Manning (2017)	95.74	94.08	–
Ma et al. (2018)	95.87	94.19	–
Fernández-G and Gómez-R (2019)	96.04	94.43	–
Zhou and Zhao (2019)(BERT)	97.00	95.43	–
Zhou, Li, and Zhao (2020) (XLNet)	97.23	95.65	–
Mrini et al. (2020) (XLNet)	97.42	96.26	–
Vinyals et al. (2015)	–	–	88.30
Zhu et al. (2013)	–	–	90.40
Vilares, Abdou, and Søgaard (2019)	–	–	91.13
Dyer et al. (2016)	–	–	91.20
Kitaev and Klein (2018)	–	–	95.13
Kitaev, Cao, and Klein (2019)	–	–	95.77
Zhou and Zhao (2019)(BERT)	–	–	95.84
Zhou, Li, and Zhao (2020) (XLNet)	–	–	96.33
Mrini et al. (2020) (XLNet)	–	–	96.38
D-MTL-AUX	94.05	92.01	91.04

Table 34: Comparison of the D-MTL-AUX models against some existing dependency and constituency models evaluated on the PTB test set.

Overall, the results show that our sequence labeling models yield good results. Even though they do not yield SOTA results, the main strength of the sequence labeling parsers is the good trade-off between speed and accuracy they provide, even when enhanced with complementary representations.

Next, we contrast our models against dependency and constituency parsers evaluated on the SPMRL treebanks. The comparison serves only for orientation purposes since the models differ with respect to the architecture (including ensemble models) and resources they rely on. Table 35 reports the accuracy for dependency parsers while Table 36 for constituency parsers.

Model	Basque	French	German	Hebrew	Hungarian	Korean	Polish	Swedish	*average*
Nivre et al. (2006)	70.11	77.98	77.81	69.97	70.15	82.06	75.63	73.21	*74.62*
Ballesteros (2013)	78.58	79.00	82.75	73.01	79.63	82.65	79.89	75.82	*78.92*
Ballesteros et al. (2015)†	78.61	81.08	84.49	72.26	76.34	86.21	78.24	74.47	*78.96*
Clergerie (2013)	77.55	82.06	84.80	73.63	75.58	81.02	82.56	77.54	*79.34*
Björkelund et al. (2013)‡	85.14	85.24	89.65	80.89	86.13	86.62	87.07	82.13	*85.36*
D-MTL-AUX	84.02	83.85	88.18	74.94	80.26	85.93	85.86	79.77	*82.85*

Table 35: Comparison of the D-MTL-AUX models against other existing dependency parsers evaluated with LAS on the SPMRL test set. The model denoted with † uses char and PoS tags, while the symbol ‡ indicates an ensemble model.

Model	Basque	French	German	Hebrew	Hungarian	Korean	Polish	Swedish	*average*
Fernández-González and Martins (2015)	85.90	78.75	78.66	88.97	88.16	79.28	91.20	82.80	*84.22*
Coavoux and Crabbé (2016)	86.24	79.91	80.15	88.69	90.51	85.10	92.96	81.74	*85.66*
Björkelund et al. (2013)‡	87.86	81.83	81.27	89.46	91.85	84.27	87.55	83.99	*86.01*
Vilares, Abdou, and Søgaard (2019)	90.85	80.40	83.42	92.05	90.38	83.24	93.93	85.54	*87.48*
Coavoux and Crabbé (2017)	88.81	82.49	85.34	89.87	92.34	86.04	93.64	84.00	*87.82*
Kitaev and Klein (2018)	89.71	84.06	87.69	90.35	92.69	86.59	93.69	84.35	*88.64*
D-MTL-AUX	90.84	80.39	82.97	91.78	90.44	82.76	93.36	85.49	*87.25*

Table 36: Comparison of the D-MTL-AUX models against other existing constituency parsers evaluated with F1 on the SPMRL test set. The symbol ‡ indicates an ensemble model.

Again, our sequence labeling parser provides competitive results compared with the ones obtained by the other systems when evaluated on several morphologically rich languages.

6.3 Conclusions

In this chapter, we have shown how our sequence labeling parser can be further enhanced with complementary data. At the core of this study lies the idea of leveraging the complementary nature of constituency and dependency paradigms and to do so, we have used a MTL hard-sharing architecture with the optional use of auxiliary losses. More concretely, we have tested models in four setups, i.e. (i) single-paradigm, single-task (S-S) (ii) single-paradigm, multi-task (S-MTL), (iii) double-paradigm, multi-task models with auxiliary losses (D-MTL-AUX) and (iv) double-paradigm, multi-task (D-MTL).

In general, the empirical results show that on average the D-MTL-AUX models outperform the baseline S-S by 0.62% of UAS for dependency parsing and by 1.14% of F1 score for constituency parsing. Furthermore, we have demonstrated that a single model that is able to encapsulate both representations also improves the baseline. Finally, we have shown that the enhanced models come at almost no cost in terms of

speed and provide a competitive performance compared to the existing systems.

7 Enhancing dependency parsing with eye-tracking data

During reading, humans pay more attention to certain parts of the sentence, suggesting that these may correspond to some core elements of the sentence (Rayner, 1998; Rayner, 2009). Therefore, it can be beneficial to guide parsers in a human-like manner which words they should attend to. In this chapter, we will focus on eye-tracking data that contains measures of eye movements during reading and we will explore how this kind of complementary data can be leveraged in sequence labeling parsing. It is especially appealing due to the fact that both eye-tracking and sequence labeling parsing work at token level and therefore can be trained within the same model.

Hence, the core idea is to use eye-tracking measures as gaze features to provide an auxiliary guidance for our parser only during training. More specifically, the gaze feature representations will be learned as auxiliary tasks in the MTL setup and thereby shared across the model. We will conduct experiments, where the models will be trained in two setups: (i) with parallel data that provides aligned dependency and gaze information and (ii) with disjoint data that includes non-overlapping treebanks, where one contains dependency trees with gaze data, while the other only dependency annotation.

7.1 Use of human language processing data in NLP

In recent years, data from various sources measuring human behavior and neural processes have become more accessible. As a result, there has been a growing interest in using recordings of the human eye and brain activity during cognitive tasks to enhance NLP, e.g. data from eye-tracking (Barrett et al., 2016; Klerke and Plank, 2019), electroencephalography (EEG) (Zhang et al., 2018) or functional magnetic resonance imaging (fMRI) (Bingel, Barrett, and Søgaard, 2016; Toneva and Wehbe, 2019). Some work has sought to explore human language processing data but still there are some uncertainties around the most optimal way of leveraging it. Recently, Hollenstein, Barrett, and Beinborn (2020) have provided some guidelines with respect to the choice of human language processing data setups and methods for human feature extraction. In this chapter, we will focus, in particular, on using

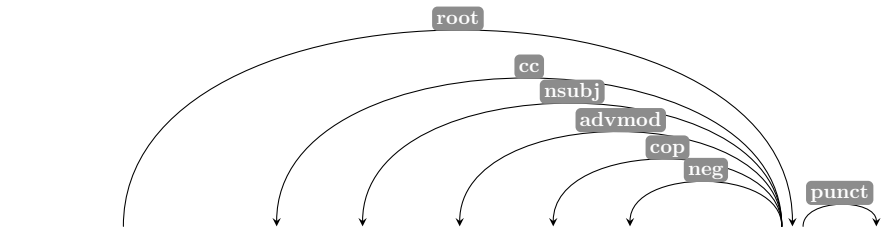
eye movement measures due to its relevance to the dependency parsing task.

7.1.1 Eye-tracking data

In general, eye-tracking data provides measures of eye movements that may reflect some cognitive processes. More concretely, one may monitor the reading process by registering eye movements called *saccades*. Between these movements, eyes also tend to remain momentarily still and this action is called *fixation*. Moreover, around $10 - 15\%$ of eye movements are *regressions*, where the saccades proceed backward (Rayner, 1998).

The eye-tracking data has been successfully applied to a range of NLP tasks, e.g. PoS tagging (Barrett et al., 2016), word embeddings evaluation (Søgaard, 2016), readability prediction (González-Garduño and Søgaard, 2017), sentiment analysis (Mishra et al., 2016) or NER (Hollenstein and Zhang, 2019). We refer the reader to Barrett (2018) for an extensive review on eye-tracking data for NLP.

Regarding the use of eye-tracking data in the context of dependency parsing, Barrett and Søgaard (2015) apply gaze features to discriminate between grammatical functions (e.g. subjects and objects) and to improve a transition-based dependency parser trained on a structured perceptron (Collins, 2002; Zhang and Nivre, 2011). However, the experiments were carried out on a small data set and with relatively low performance. More recently, Lopopolo et al. (2019) explore the reversed reliance i.e. whether edges of relations produced by a dependency parser may match backward saccades. They conclude that the results are in line with the view that eye movements in fact may reflect some syntactic structure.



`first fix.dur.`	0	0	187	176	176	0	173	0
n `fix.`	0	0	1	1	1	0	1	0
`first pass dur.`	0	0	187	176	176	0	173	0

Figure 29: An example of a dependency tree with corresponding eye movement measures extracted from the Dundee treebank.

7.1.2 Leverage of gaze features

The simplest way of leveraging eye-tracking data in dependency parsing is to use the measures of eye movements that are aligned with a dependency tree. An example of a dependency tree with gaze measures is shown in Figure 29. In general, some particular gaze measures are selected to be used in a NLP task. For instance, Barrett et al. (2016) employ 22 gaze features for experiments on PoS tagging, while Hollenstein and Zhang (2019) make use of 17 features for NER. In our approach, based on an empirical search, we select 12 gaze features that are detailed in Table 37. We follow Barrett et al. (2016) and Hollenstein and Zhang (2019) and group them into the following subsets:

1. BASIC group includes features at the word level.
2. EARLY measures reflect early syntactic processing and lexical access.
3. LATE measures reflect late syntactic processing and disambiguation.
4. CONTEXT group takes into account measures of the surrounding tokens. In our approach, we restrict this group to the previous and next words of w.

Gaze features		Description
BASIC	`total fix.dur.`	total fixation duration on a word w
	`mean fix.dur.`	mean of all fixation durations on w
	n `fix.`	total number of fixations on w
	`fix.prob.`	fixation probability on w
EARLY	`first fix.dur.`	first fixation duration on w
	`first pass dur.`	all fixation durations during the first pass
LATE	n `re-fix`	number of times w being fixated again
	`re-read prob.`	probability of w being read again
CONTEXT	$w-1$ `fix.dur.`	fixation duration on the preceding word
	$w+1$ `fix.dur.`	fixation duration on the next word
	$w-1$ `fix.prob.`	fixation probability on the preceding word
	$w+1$ `fix.prob.`	fixation probability on the next word

Table 37: The selected gaze features for dependency parsing as sequence labeling based on the feature grouping following Barrett et al. (2016).

Various work, that has employed eye-tracking data to NLP, differs in methods for leveraging the gaze features in their models. One of the challenges that emerges is that limited or no eye-tracking data is available at inference time. In previous studies (Barrett and Søgaard, 2015;

Barrett et al., 2016), it has been suggested that real-time eye-tracking data may be widely accessible in the near future due to the fact that a cheap eye-tracking equipment may become ubiquitous in webcams or smartphones. Although, this seems to be far from guaranteed and raises privacy concerns (Liebling and Preibusch, 2014). Hence, leveraging eye-tracking data as *token-level gaze features* may not be practical in real-world setups, since they are required both during training and testing.

To alleviate the need of eye-tracking data at inference time, other methods have been proposed (Barrett and Søgaard, 2015; Hollenstein and Zhang, 2019) that consist in deriving the gaze features only from training data. An example of that are *type-level aggregated gaze features* that are retrieved from the vocabulary in the training set. More specifically, after being aggregated they are contained in a lookup table and at test time if an input token matches the entry of the lexicon, the pre-computed gaze values are returned, otherwise the value for that token will be set up to unknown. It has been shown that models using the type-level aggregated gaze features perform superiorly to the ones relying on the token-level features (Barrett et al., 2016; Hollenstein and Zhang, 2019). However, this approach poses another challenge. Namely, this type of gaze features is built based on the vocabulary seen during training, hence they may not be available for all tokens at testing time. In fact, Hollenstein and Zhang (2019) find out in their study on NER that 41.09% of the tokens remained unknown at inference time. To our best knowledge, this method has not been applied to syntactic parsing.

In an alternative approach, the features are learnt by the model, such that they have a direct impact on its weights or the acquired representations (for a more complete overview we refer to Hollenstein, Barrett, and Beinborn (2020)). For instance, Barrett et al. (2018) use the eye-tracking data to regularize attention weights in a recurrent neural network for sequence classification. This method additionally enables training a model with disjoint data, where eye-tracking data does not need to overlap with the data for the target task. Furthermore, the gaze features have been successfully leveraged in the MTL setup for sentence compression and syntactic tagging tasks, where the gaze predictions are treated as auxiliary tasks (Klerke, Goldberg, and Søgaard, 2016; Klerke and Plank, 2019).

7.2 Models and experiments

In our study, we will perform experiments in two data setups that will be introduced below. Then we will show how gaze features can be leveraged in our sequence labeling framework. Finally, we will analyze and discuss the empirical results of the models enhanced with the gaze features.

7.2.1 Parallel and disjoint data

We will make use of two data setups. The first one provides *parallel* training data, where dependency and gaze annotations are aligned, while the second one consists of *disjoint*, non-overlapping treebanks, where one of them contains the aligned gaze annotation, while the second only dependency trees. We motivate the use of disjoint treebanks by the fact that when applying gaze features, in particular, to dependency parsing, it is preferable to be able to train them with an arbitrary dependency treebank that usually does not contain any gaze annotation.

Parallel data The Dundee Treebank (Barrett, Agić, and Søgaard, 2015) will serve as the parallel training data. It is based on the English Dundee Corpus (Kennedy, Hill, and Pynte, 2003) that provides recordings of eye movements of ten English-speaking participants reading 20 newspaper articles from *The Independent*. The Dundee Treebank enhances the Dundee Corpus with a syntactic annotation according to the UD guidelines and thereby enables a straightforward use of eye-tracking measures in dependency parsing.

We split the data into training, development and test set $(80 - 10 - 10)$, respectively. The sentences were priorly randomly shuffled and we ensure that the same sentence coming from the ten participants is assigned to the same set. Moreover, 40 sentences (4 unique sentences read by the ten participants) were removed due to incoherence in their dependency annotation (we observe that some of the dependency trees were ill-formed), resulting in 23640 sentences in total (2364 unique sentences).

Disjoint data We use two treebanks as our disjoint training data, namely PTB and the Dundee Treebank. The former is a commonly used English dependency treebank that does not provide any gaze information (see a more detailed description in Section 2.2.2). Hence, we train our model on the gaze annotations of the Dundee Treebank and also on the trees of the PTB. We use a data split of $90 - 10 - 0$ for the Dundee Treebank, since solely the PTB test set is used at the test time.

7.2.2 Gaze-leveraged sequence labeling parsing

To amplify the benefits of our approach and facilitate the use of gaze features, we do not require that the gaze information is available at inference time. The core idea is to inject during training the information about the human eye movements to the model, so that the acquired gaze representations help to guide the dependency parser at testing time. In our experiments, the gaze features are learned with a BiLSTM-based architecture within the MTL setup, where dependency

parsing is defined as the main task, while the gaze features are framed as the auxiliary tasks. The architecture is illustrated in Figure 30. The dependency and gaze representations are first shared across the model followed by separate feed-forward layers with *softmaxes* for predicting labels for each subtask.

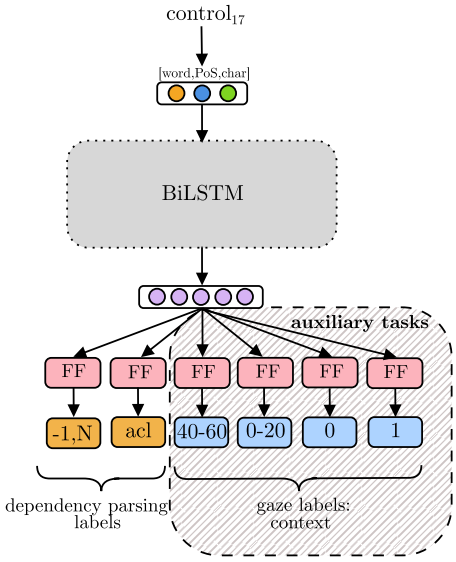


Figure 30: Architecture for learning gaze features in the MTL setup.

For this chapter, we rely on the relative PoS-based encoding which, as explained in previous chapters, can be learned as two tasks that consist in: (i) predicting the relative PoS-based position of the head (o_i, p_i) and (ii) predicting the dependency relation (d_i). In the case of eye-tracking data, we select 12 gaze features divided into four groups as detailed in Table 37. We leverage them in a twofold manner: (i) only *one* of the gaze features is learned as an auxiliary task and (ii) gaze features belonging to the same *group* are learned as multiple auxiliary tasks. It is also worth mentioning that following Hollenstein and Zhang (2019), we discretize some gaze features. Namely, `total fix.dur.`, `mean fix.dur.`, `first fix.dur.`, `first pass dur.`, $w-1$ and $w+1$ `fix.dur.` obtain values as percentile intervals with a bin size of 20, while the remaining features keep their raw values.

In the parallel setup, the cross-entropy loss is computed as $\mathcal{L} = \mathcal{L}_{(o,p)} + \mathcal{L}_d + \sum_a \beta_a \mathcal{L}_a$, where losses coming from the label predictions for dependency parsing and the weighted losses from each gaze feature prediction are added. In the disjoint setup, in turn, for each batch we randomly pick all samples coming either from the dependency parsing treebank (PTB) or from the corpus with eye-tracking data (The Dundee Treebank). More concretely, we pick samples that have not been taken yet from one of the treebanks and the loss is back-propagated for the

	Gaze features	dev set		test set	
		UAS	LAS	UAS	LAS
	baseline	85.36	79.40	84.37	78.24
BASIC	`total fix.dur.`	85.34	79.35	84.06	77.44
	`mean fix.dur.`	85.21	79.38	84.59	**78.70**
	n `fix.`	85.32	79.29	83.71	77.57
	`fix.prob.`	85.32	79.57	84.33	77.91
	`basic feats aux.`	85.36	79.57	83.86	77.75
EARLY	`first fix.dur.`	85.30	79.46	**84.64**	78.57
	`first pass dur.`	85.50	79.49	84.55	78.39
	`early feats aux.`	**85.61**	79.57	84.37	78.11
LATE	n `re-fix.`	85.52	79.25	83.86	77.91
	`re-read prob.`	85.34	79.57	83.86	77.37
	`late feats aux.`	85.54	79.64	84.10	77.73
CONTEXT	$w-1$ `fix.prob.`	85.17	79.47	84.26	77.93
	$w+1$ `fix.prob.`	85.36	79.07	84.24	78.06
	$w-1$ `fix.dur.`	85.43	79.68	84.50	77.95
	$w+1$ `fix.dur.`	85.39	79.53	**84.64**	78.30
	`context feats aux.`	**85.61**	**79.72**	84.33	78.24

Table 38: Performance of models in the parallel setup evaluated on the Dundee treebank. The baseline models do not use any gaze features as auxiliary task(s).

output labels associated to a given treebank. More formally, we define this loss as: $\mathcal{L} = \tau(\mathcal{L}_{(o,p)} + \mathcal{L}_d) + (1-\tau)\sum \beta_{aux}\mathcal{L}_{aux}$, where τ is a binary flag with 1, when the batch holds samples from the dependency treebank and 0, otherwise. The first term stands for the losses coming from dependency labels, while the second includes the weighted losses from each gaze feature label. Each of the auxiliary tasks has a weighting factor of 0.1. The hyperparameters are detailed in Section A.1.1.

7.2.3 Experiments

Now, we move on to the experiments in order to explore the viability of our method in terms of accuracy, as well as, to determine which gaze features contribute most to the improvements of our sequence labeling parser. We will first examine the performance of the models that rely on the parallel training data and then on the disjoint data.

Experiment on parallel data We will test the performance of our parser trained on the Dundee Treebank that provides both dependency and gaze annotations. We compare models that use gaze features as auxiliary tasks against the baseline model that only learns dependency

Gaze features		dev set		test set	
		UAS	LAS	UAS	LAS
	baseline	93.98	91.67	93.86	91.80
BASIC	`total fix.dur.`	93.94	91.60	93.99	91.92
	`mean fix.dur.`	**94.12**	**91.84**	93.95	91.82
	n `fix.`	93.97	91.70	93.91	91.87
	`fix.prob.`	93.98	91.71	93.99	91.93
	`basic feats aux.`	94.00	91.69	93.84	91.81
EARLY	`first fix.dur.`	94.07	91.81	93.87	91.80
	`first pass dur.`	93.93	91.58	93.79	91.70
	`early feats aux.`	94.04	91.78	93.96	91.88
LATE	n `re-fix.`	94.01	91.69	93.87	91.79
	`re-read prob.`	94.03	91.74	93.98	91.89
	`late feats aux.`	93.98	91.58	93.92	91.90
CONTEXT	$w-1$ `fix.prob.`	94.02	91.65	93.95	91.93
	$w+1$ `fix.prob.`	93.88	91.61	93.89	91.82
	$w-1$ `fix.dur.`	94.06	91.65	93.86	91.83
	$w+1$ `fix.dur.`	93.91	91.69	93.89	91.84
	`context feats aux`	93.93	91.63	**94.01**	**91.98**

Table 39: Performance of models in the disjoint setup evaluated on the PTB treebank. The gaze features were learned as auxiliary task(s) from the disjoint data set (the Dundee treebank).

labels and does not employ any eye-tracking features. The results of the models evaluated on the Dundee Treebank are shown in Table 38.

In general, the results suggest that the models leverage the respective gaze features to a varying degree. Particularly, it is prominent that the models using different gaze features produce inconsistent results with respect to the development and test set. For instance, the evaluation on the dev set shows that the features that lead to the greatest improvements are those coming from the *early* and *context* group learned as multiple auxiliary tasks (`early` and `context feats aux.`). On the test set, in turn, `mean fix.dur.` and `first fix.dur.` provide the best parsing results, where the former improves LAS by +0.46% and the latter by +0.33%. However, it is worth noting that the Dundee treebank is relatively small (∼2k sentences) and thereby it is difficult to generalize based on the results. Hence, now we move to examine models on a larger dependency treebank.

Experiment on disjoint data In this setup, we perform experiments with models that are trained on non-overlapping treebanks. Namely, the Dundee treebank is used to learn both gaze features and the aligned dependencies, while the PTB treebank provides solely depen-

dency annotations. However, the models are evaluated on a non-gazed data set: the PTB treebank. The results are shown in Table 39.

Overall, the results in the disjoint setup indicate that the gains of using gaze features diminish in comparison with the parallel setup. When evaluating the parser on the dev set, `mean fix.dur.` and `first fix.dur.` lead to the greatest improvements in the LAS score with +0.17% and +0.14%, respectively. On the test set, in turn, features coming from the context group learned as multiple auxiliary tasks improve LAS by +0.18%, followed by `fix.prob.` and $w-1$ `fix.prob.` with +0.13%, `total fix.dur.` with +0.12% and `late feats aux.` with +0.10%. The lower gains in this setup can be partly explained by the fact that PTB outsizes the Dundee Treebank, hence the gaze signal may be weaker. Perhaps in the disjoint setup, the transfer of representations coming from non-overlapping treebanks is hindered by differences in vocabulary and the context they appear in. On the other hand, the results seem to be more consistent which may be triggered by using a larger and more representative treebank.

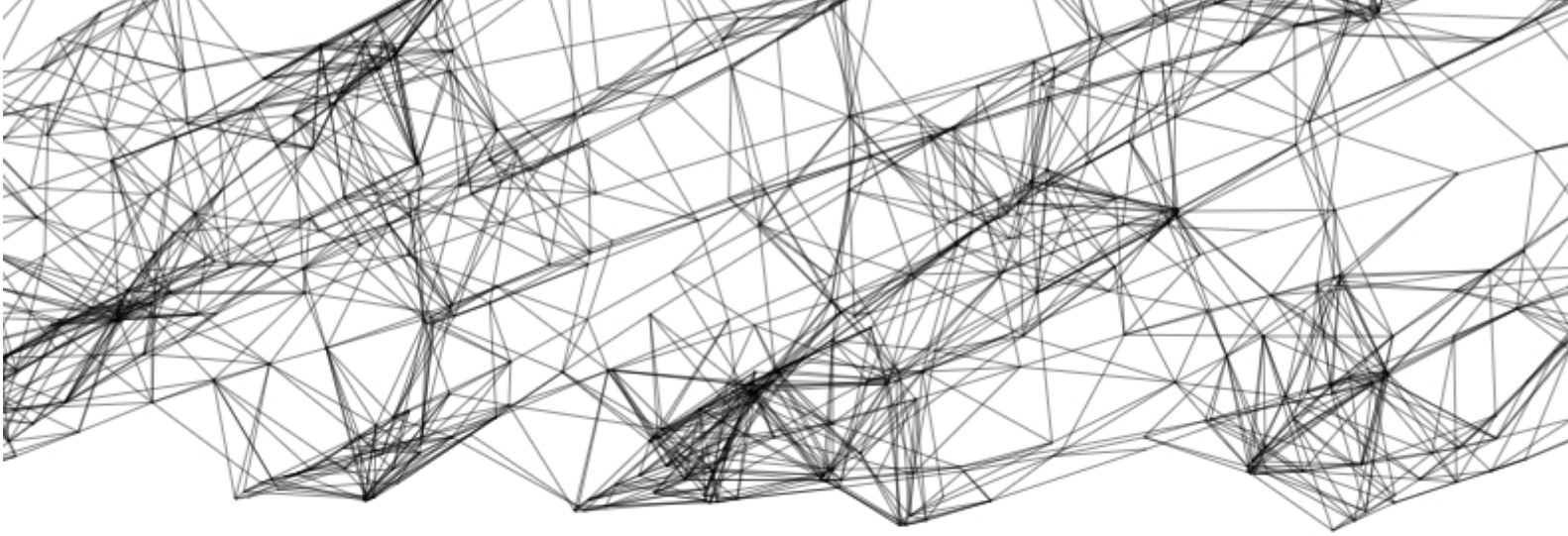
7.3 Conclusions

In this chapter, we have proposed to leverage eye-tracking data in dependency parsing within the sequence labeling framework. Specifically, we have explored the use of eye movement measures that reflect some cognitive processes to guide our dependency parser with some sort of human gaze attention on particular words. Unlike some previous work, we do not rely on the assumption that the eye-tracking data is available at inference time. In our approach, the gaze features are learned as auxiliary tasks in the MTL setup with parallel and disjoint training data.

The results have shown that models enhanced with the gaze feature representations obtain modest but positive results providing some initial attempts in leveraging eye-tracking data in the context of dependency parsing. The evaluation of the models suggests, however, that grouping and treating gaze features as auxiliary tasks may enhance the model learning, although the gains remain unstable.

Taken together, our approach offers many advantages. For instance, the gaze features are included in the models, hence they are not required during testing facilitating the use of arbitrary treebanks at the test time. Moreover, due to the architecture any other types of complementary data can be easily applied in our sequence labeling framework, i.e. NER or chunking data. No less important is the fact that our parser provides competitive parsing speed, even when enhanced with complementary data. Still, there is a room for improvements, for instance in increasing the generalization capabilities of our model to mitigate the divergence in the results between the dev and test set. To do so, one may consider applying other architecture in our framework, e.g. following Barrett

et al. (2018) eye-tracking data approach could serve to regularize the attention. However, that would come at some cost to speed.



Part IV

CLOSING REMARKS

8 Conclusions

8.1 Discussion and limitations

In this thesis, we have recast dependency parsing as sequence labeling as an alternative approach to the canonical transition- and graph-based methods. As stated in the motivation in PART I, one of the aims of our work was to provide a fast and accurate system that (i) relies on generic methods, such that it can be applied using any sequence labeling software and (ii) can be easily integrated with other downstream tasks facilitating the use of full parse trees as features. To do so, we enhanced the primary study by Spoustová and Spousta (2010) and made this technique practical in the neural network settings. Our work was also motivated by the study of Gómez-Rodríguez and Vilares (2018) that demonstrated that continuous constituent parsing could be effectively reduced to a sequence labeling task when training contextualized neural models (e.g. LSTMs), but not traditional machine learning systems, such CRF or perceptrons.

In general, our sequence labeling parsers have shown to maintain a good speed/accuracy trade-off with a conceptually simple architecture that alleviates the recurring bottleneck of parsing speed. Specifically, a sequence labeling parser foregoes the need of traditional parsing algorithms or auxiliary structures by replacing them with linearization methods for transforming a dependency tree into a sequence of labels. This linearization enables parsing over a sentence of length n into exactly n tagging actions resulting in fast parsing. Furthermore, we have explored the leverage of external complementary data by a sequence labeling parser. It can be especially useful, for instance, in a low-resource setup, since it facilitates using additional data that can be learned together in the sequence labeling setups. In particular, we have shown that our model is capable of exploiting representations coming from another parsing paradigm, as well as, from human gaze data by means of MTL.

In PART II, we have introduced three families of encodings that are the backbone for dependency parsing as sequence labeling, i.e. head selection encodings (Chapter 3), bracketing-based encodings (Chapter 4) and transition-based encodings (Chapter 5). The main distinction between the encoding families is the linearization methods they apply. More concretely, encoding of dependency trees can be defined as a problem of (i) head selection, (ii) matching well-balanced bracketing elements and (iii) splitting transitions retrieved from transition-based

systems into subsequences, such that they can be assigned to each token. In Figure 31 we provide an outline of the proposed encodings with respect to their family hierarchy. For each encoding, we have provided a formalisation, followed by a description of the decoding method that is required in order to transform the predicted sequence of labels into an output tree. Finally, we have discussed their strengths and limitations.

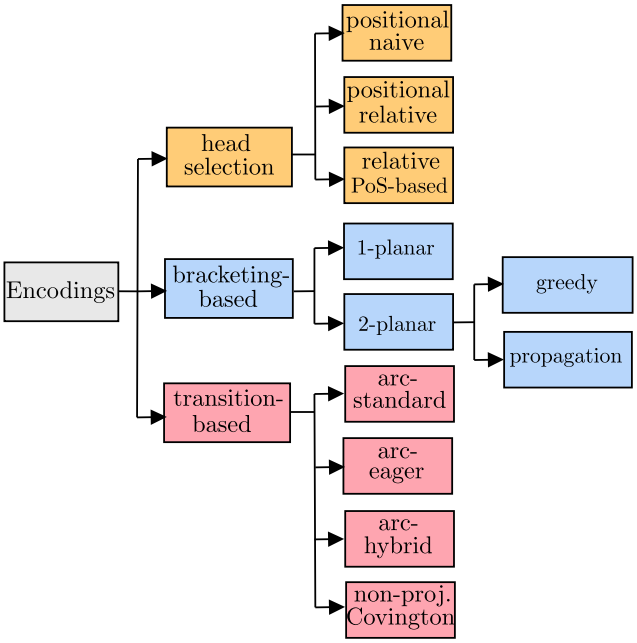


Figure 31: Encoding hierarchy outline.

In Chapter 3, we have presented the first encoding family based on a head selection criterion, whose main trait is that it encodes the absolute position of the head or the relative distance from the dependent to the head. More specifically, we distinguish (i) the naive positional encoding, in which each label holds information about the index position of the head, (ii) the relative positional encoding, which captures the difference between the index position of the head and the dependent and (iii) the relative PoS-based encoding, in which each label holds the relative distance from the dependent to the head considering only words with the head's PoS tag. It is worth mentioning that this family can be used to encode the head according to any word property, for instance, as in the work of Lacroix (2019). We have carried out a number of experiments that revealed several characteristics of each head selection encoding. First of all, the encodings differ in the label space they generate. It is prominent that the naive positional encoding suffers the most from label sparsity, which is reflected in a considerably lower accuracy. The remaining encodings, in turn, provide to a greater extent more compact tree representations. This can be explained by the fact that the naive encoding needs to generate a label for each index position, which can be compressed more efficiently using the relative positions instead. The

results have also shown that the relative PoS-based encoding yields the best accuracy, hence we considered it as our baseline for comparison against other encoding families. However, it is worth noting that this encoding has a limitation that needs to be considered. Namely, in a more real-world setup it requires an additional step of computing PoS tags, which as a result increases the model latency. This also implies that the performance of this encoding is conditioned on the quality and quantity of the predicted PoS tags since it may vary across treebanks. It also contradicts with the recent work that questioned the utility of PoS tags in dependency parsing (Lhoneux et al., 2017; Anderson and Gómez-Rodríguez, 2020b).

In Chapter 4, we have presented the bracketing-based family. First, we introduced the relaxed 1-planar bracketing-based encoding, where left and right dependency arcs are represented in terms of balanced pairs of bracketing elements (`<,\` and `/,>`, respectively). The evidence from our study has proved that this encoding is viable and compact in terms of the generated labels, although it obtains a slightly lower accuracy than the relative PoS-based encoding (when PoS tags are used as input features to the model). Additionally, it has a limitation with respect to the coverage of non-projective arcs. Specifically, the relaxed 1-planar encoding might be lossy, since it is able to only encode non-projective arcs in the opposite directions and, as a consequence, crossing arcs in the same direction are decoded as projective ones. Unlike the head selection encodings that are capable of fully encoding non-projective trees, the relaxed 1-planar bracketing-based one needs to be extended in order to handle non-projectivity. To overcome this shortcoming, we have proposed a variant that makes use of the property called 2-planarity. Briefly, non-projectivity may be preserved almost entirely by partitioning the arcs into two planes, such that arcs in the same plane do not cross. In the context of dependency parsing as sequence labeling, it can be applied using additional bracketing elements denoting that they belong to the second plane ($<^*, \backslash^*$ and $/^*, >^*$). In order to assign arcs to one of the two planes, we have defined two strategies: a greedy one that averts from using the second plane and one based on restriction propagation on the crossings graph. The latter has a theoretical advantage of guaranteeing a full coverage of 2-planar trees. The empirical results have shown that the 2-planar bracketing-based encodings are able to almost fully cover non-projective arcs, which is also reflected in the improved accuracy on highly non-projective treebanks. We have found that the 2-planar encodings based on different plane assignment strategies perform on par in terms of accuracy, even though the strategy based on restriction propagation has a theoretical advantage. Presumably, this advantage did not translate into a greater accuracy due to the fact that some dependency trees possibly were not 2-planar, hence the theoretical advantage could not be applied. The experiments have shown that the 2-planar variants

Encoding	supports non-proj?	external feats?	excessive label set size?
naive-positional	✓	✗	✓
rel-positional	✓	✗	✗
rel-PoS	✓	✓	✗
1p-brackets	✓/✗	✗	✗
2p-greedy	✓	✗	✗
2p-prop	✓	✗	✗
arc-standard	✗	✗	✗
arc-eager	✗	✗	✗
arc-hybrid	✗	✗	✗
Covington	✓	✗	✓

Table 40: Encodings' characteristics in terms of supporting non-projectivity, requiring external features (e.g., computing PoS tags) and generating an excessive output vocabulary size.

supporting non-projectivity within the sequence labeling approach are feasible and come at almost no additional cost to parsing speed.

In Chapter 5, we have proposed another encoding family that is based on the left-to-right transition-based systems. This approach establishes a theoretical link between the transition- and sequence labeling paradigms. More concretely, we have devised a theoretical framework to map transition-based parsers to sequence labeling parsers using the concept of read transitions to split the transition sequences. Thereby, such subsequences can be then assigned as labels to each input word. We have empirically tested our approach by obtaining four transition-based encodings based on the arc-standard, arc-eager, arc-hybrid and non-projective Covington algorithms. However, it is worth mentioning that the proposed method is generic and applicable to a wide range of left-to-right transition-based systems, also beyond dependency parsing. In general, our study has shown that the transition-based encodings are viable both in terms of the accuracy and parsing speed. The advantage of this method is that new encodings can be retrieved automatically from the existing transition-based algorithms. Moreover, unlike a transition-based parser, the sequence labeling parser with the transition-based encodings does not rely on stack-based features. Although, we observed that a weak point of systems that do not run in $\mathcal{O}(n)$ (such as the non-projective Covington) is that they may result in an extensive output vocabulary precluding an efficient learning.

Taken together, the families of encodings differ in how they embed and represent dependency trees as labels. As a result, they are associated with miscellaneous facets (whether they handle non-projectivity, make use of external features, such as PoS tags or generate a large label space) that need to be considered when trying to select the most suitable encoding for an arbitrary setup. In Table 40, we provide an overview of the facets with respect to each encoding. Regarding ac-

curacy, the relative PoS-based encoding obtains the best performance across all encodings, but solely in the case of using PoS tags as input parameters. Since this encoding strongly depends on the PoS tags, its performance may deteriorate due to their low quality, an inadequate level of granularity or in the case they are only used for decoding (not as input features). Hence, other encodings can be a better alternative if the additional step of computing PoS tags in not preferable, for instance due to the chosen architecture (such as BERT) that does not require them or due to the intention of maintaining a low model latency. If so, the bracketing-based encodings yield the best results. Although, the transition-based encodings also provide comparable results (except for the non-projective Covington). When parsing highly non-projective treebanks, the head selection encodings are able to fully support crossing arcs, while the 2-planar variants of the bracketing-based encoding almost completely are able to preserve them. The non-projective Covington, even though it supports non-projectivity, suffers from the excessive output vocabulary resulting in lower accuracy, while the remaining transition-based encodings only cover projective trees.

In PART III we have covered enhanced training of our sequence labeling parser with complementary representations retrieved from external data. More concretely, we aimed at improving the performance of the parser by leveraging representations from constituency parsing and eye-tracking data by means of auxiliary tasks in the MTL setup.

In Chapter 6, we have explored whether the leverage of the complementary nature of dependency and constituency paradigms can be facilitated with a sequence labeling parser in contrast to the previous approaches that were more complex. We have tested the feasibility of this method by learning the representations of the counterpart abstractions as auxiliary tasks. The results have shown that this method consistently improves the performance of the paradigm of interest. Furthermore, we have demonstrated that dependency and constituency parsing can be performed by a single model that learns both paradigms as main tasks at almost no cost in terms of speed.

In Chapter 7, we have investigated the impact of learning another kind of complementary data, namely eye-tracking data, where at the core lies the idea of guiding our sequence labeling parser with measures of eye movements. To do so, gaze features were learned as auxiliary tasks requiring the eye-tracking data only for training, which is practical since the human language processing data still is not easily accessible and collecting it on a large scale raises privacy concerns (Liebling and Preibusch, 2014). Moreover, we have examined our method in two data setups: (i) with parallel training data, in which dependency and gaze annotation are aligned and (ii) disjoint training data, which in addition contains a non-overlapping treebank with solely dependency trees. Overall, the empirical results are modest but positive suggesting that learning gaze features as auxiliary tasks may be beneficial for

sequence labeling parsers. However, based on the results one may not clearly determine which group or individual features guide the parser best, since the gains are unstable across the dev and test set, as well as, between the different data setups. Nevertheless, the findings encourage further investigation on the use of eye-tracking data in dependency parsing.

8.2 Future work

This work presents the first approaches enhancing dependency parsing as a sequence labeling problem, and that have achieved practical results in terms of speed and accuracy. Our study has revealed that dependency parsing as sequence labeling is viable and fast. Yet, it still faces some challenges and there is room for improvement. For instance, in the context of recent advances of deep neural networks, future work needs to be carried out to establish whether other neural architectures beyond BiLSTMs and BERT may boost the performance of sequence labeling parsers. For instance, Vilares and Gómez-Rodríguez (2020) have recently applied vanilla transformers and DistilBERT to discontinuous constituency parsing as sequence labeling. Furthermore, since one of the main motivations of recasting dependency parsing as sequence labeling is the competitive parsing speed, one may aim at proposing a model based on some other architecture that yields even better speed up. For instance, in future work the teacher-student distillation technique may be explored in the context of sequence labeling parsing, where a pretrained language model (e.g. BERT) could be used as the teacher and a single LSTM as the student (or, even more extreme, a feed-forward network with very efficient feature extraction).

Regarding the linearization methods, further investigations should not be limited to the ones proposed in this work. Regarding the head selection encodings, one may apply any alternative method to encode the heads (for instance, see the encoding proposed by Lacroix (2019)). In the relative PoS-based encoding, other properties than PoS tags may be used for encoding, e.g. morphological features. With respect to the 2-planar bracketing-based encoding, we have proposed two plane assignment strategies that minimize the use of the second plane. Although, one may explore other strategies based on different criteria. For instance, in a similar vein, Anderson and Gómez-Rodríguez (2021) explore some linguistic criteria for splitting EUD graphs into trees. When it comes to the transition-based encodings, one may explore other existing left-to-right transition-based systems to retrieve additional encodings beyond these included in the experiments in this thesis. Furthermore, the future work should not be limited to exploring encodings derived solely from dependency parsers, since our mapping method is generic and can be also applied to constituent or semantic parsers.

When it comes to the use of complementary data, future research should be undertaken in order to establish methods for the most optimal leverage of such data in dependency parsing beyond the use of auxiliary tasks in the MTL setup. For instance, more research is needed for determining how to leverage data that is only available during training and in the disjoint data setup, since there is a wide range of cognitive measures that may be beneficial for dependency parsing. Furthermore, it might be interesting to explore other sources, for instance, with visual information (Salama and Menzel, 2016) or audio, to direct the focus on multi-modal dependency parsing as sequence labeling.

Bibliography

Anderson, Mark and Carlos Gómez-Rodríguez (July 2020a). «Distilling Neural Networks for Greener and Faster Dependency Parsing.» In: *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*. Online: Association for Computational Linguistics, pp. 2–13. DOI: 10.18653/v1/2020.iwpt-1.2. URL: https://www.aclweb.org/anthology/2020.iwpt-1.2.

— (Nov. 2020b). «On the Frailty of Universal POS Tags for Neural UD Parsers.» In: *Proceedings of the 24th Conference on Computational Natural Language Learning*. Online: Association for Computational Linguistics, pp. 69–96. DOI: 10.18653/v1/2020.conll-1.6. URL: https://www.aclweb.org/anthology/2020.conll-1.6.

— (2021). «Splitting EUD graphs into trees: A quick and clatty approach.» In: *arXiv preprint arXiv:2106.13155*.

Attardi, Giuseppe (June 2006). «Experiments with a Multilanguage Non-Projective Dependency Parser.» In: *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*. New York City: Association for Computational Linguistics, pp. 166–170. URL: https://www.aclweb.org/anthology/W06-2922.

Ballesteros, Miguel (Oct. 2013). «Effective Morphological Feature Selection with MaltOptimizer at the SPMRL 2013 Shared Task.» In: *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*. Seattle, Washington, USA: Association for Computational Linguistics, pp. 63–70. URL: https://www.aclweb.org/anthology/W13-4907.

Ballesteros, Miguel and Xavier Carreras (July 2015). «Transition-based Spinal Parsing.» In: *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*. Beijing, China: Association for Computational Linguistics, pp. 289–299. DOI: 10.18653/v1/K15-1029. URL: https://www.aclweb.org/anthology/K15-1029.

Ballesteros, Miguel, Chris Dyer, and Noah A. Smith (Sept. 2015). «Improved Transition-based Parsing by Modeling Characters instead of Words with LSTMs.» In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 349–359. DOI: 10.18653/v1/D15-1041. URL: https://www.aclweb.org/anthology/D15-1041.

Ballesteros, Miguel et al. (Nov. 2016). «Training with Exploration Improves a Greedy Stack LSTM Parser.» In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, pp. 2005–2010. DOI: 10.18653/v1/D16-1211. URL: https://www.aclweb.org/anthology/D16-1211.

Barrett, Maria (Oct. 2018). «Improving natural language processing with human data: Eye tracking and other data sources reflecting cognitive text processing.» English. PhD thesis. Denmark.

Barrett, Maria, Željko Agić, and Anders Søgaard (2015). «The Dundee treebank.» In: *The 14th International Workshop on Treebanks and Linguistic Theories (TLT 14)*.

Barrett, Maria and Anders Søgaard (Sept. 2015). «Using reading behavior to predict grammatical functions.» In: *Proceedings of the Sixth Workshop on Cognitive Aspects of Computational Language Learning*. Lisbon, Portugal: Association for Computational Linguistics, pp. 1–5. DOI: 10.18653/v1/W15-2401. URL: https://www.aclweb.org/anthology/W15-2401.

Barrett, Maria et al. (Aug. 2016). «Weakly Supervised Part-of-speech Tagging Using Eye-tracking Data.» In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 579–584. DOI: 10.18653/v1/P16-2094. URL: https://www.aclweb.org/anthology/P16-2094.

Barrett, Maria et al. (Oct. 2018). «Sequence Classification with Human Attention.» In: *Proceedings of the 22nd Conference on Computational Natural Language Learning*. Brussels, Belgium: Association for Computational Linguistics, pp. 302–312. DOI: 10.18653/v1/K18-1030. URL: https://www.aclweb.org/anthology/K18-1030.

Bingel, Joachim, Maria Barrett, and Anders Søgaard (Aug. 2016). «Extracting token-level signals of syntactic processing from fMRI - with an application to PoS induction.» In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 747–755. DOI: 10.18653/v1/P16-1071. URL: https://aclanthology.org/P16-1071.

Bingel, Joachim and Anders Søgaard (Apr. 2017). «Identifying beneficial task relations for multi-task learning in deep neural networks.» In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Valencia, Spain: Association for Computational Linguistics, pp. 164–169. URL: https://www.aclweb.org/anthology/E17-2026.

Björkelund, Anders et al. (Oct. 2013). «(Re)ranking Meets Morphosyntax: State-of-the-art Results from the SPMRL 2013 Shared Task.» In: *Proceedings of the Fourth Workshop on Statistical Parsing of*

Morphologically-Rich Languages. Seattle, Washington, USA: Association for Computational Linguistics, pp. 135–145. URL: https://www.aclweb.org/anthology/W13-4916.

Bojanowski, Piotr et al. (2017). «Enriching Word Vectors with Subword Information.» In: *Transactions of the Association for Computational Linguistics* 5, pp. 135–146. DOI: 10.1162/tacl_a_00051. URL: https://www.aclweb.org/anthology/Q17-1010.

Buchholz, Sabine and Erwin Marsi (June 2006). «CoNLL-X Shared Task on Multilingual Dependency Parsing.» In: *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*. New York City: Association for Computational Linguistics, pp. 149–164. URL: https://www.aclweb.org/anthology/W06-2920.

Cao, Qingxing et al. (2021). «Interpretable Visual Question Answering by Reasoning on Dependency Trees.» In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.3, pp. 887–901. DOI: 10.1109/TPAMI.2019.2943456.

Caruana, Rich (1997). «Multitask learning.» In: *Machine learning* 28.1, pp. 41–75.

Changpinyo, Soravit, Hexiang Hu, and Fei Sha (Aug. 2018). «Multi-Task Learning for Sequence Tagging: An Empirical Study.» In: *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, pp. 2965–2977. URL: https://www.aclweb.org/anthology/C18-1251.

Chen, Danqi and Christopher Manning (Oct. 2014). «A Fast and Accurate Dependency Parser using Neural Networks.» In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 740–750. DOI: 10.3115/v1/D14-1082. URL: https://www.aclweb.org/anthology/D14-1082.

Choi, Jinho D. and Andrew McCallum (Aug. 2013). «Transition-based Dependency Parsing with Selectional Branching.» In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 1052–1062. URL: https://www.aclweb.org/anthology/P13-1104.

Choi, Jinho D. and Martha Palmer (June 2011). «Getting the Most out of Transition-based Dependency Parsing.» In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, pp. 687–692. URL: https://www.aclweb.org/anthology/P11-2121.

Chomsky, Noam (1956). «Three models for the description of language.» In: *IRE Transactions on information theory* 2.3, pp. 113–124.

Clergerie, Éric de la (Oct. 2013). «Exploring beam-based shift-reduce dependency parsing with DyALog: Results from the SPMRL 2013 shared task.» In: *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*. Seattle, Washington, USA: Association for Computational Linguistics, pp. 53–62. URL: https://www.aclweb.org/anthology/W13-4906.

Coavoux, Maximin and Benoît Crabbé (Aug. 2016). «Neural Greedy Constituent Parsing with Dynamic Oracles.» In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 172–182. DOI: 10.18653/v1/P16-1017. URL: https://www.aclweb.org/anthology/P16-1017.

— (Apr. 2017). «Multilingual Lexicalized Constituency Parsing with Word-Level Auxiliary Tasks.» In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Valencia, Spain: Association for Computational Linguistics, pp. 331–336. URL: https://www.aclweb.org/anthology/E17-2053.

Collins, Michael (July 2002). «Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms.» In: *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*. Association for Computational Linguistics, pp. 1–8. DOI: 10.3115/1118693.1118694. URL: https://www.aclweb.org/anthology/W02-1001.

Collobert, Ronan et al. (2011). «Natural Language Processing (Almost) from Scratch.» In: *Journal of Machine Learning Research* 12.76, pp. 2493–2537. URL: http://jmlr.org/papers/v12/collobert11a.html.

Covington, Michael A (2001). «A fundamental algorithm for dependency parsing.» In: *Proceedings of the 39th annual ACM southeast conference*. Citeseer, pp. 95–102.

Devlin, Jacob et al. (June 2019). «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.» In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://www.aclweb.org/anthology/N19-1423.

Dozat, Timothy and Christopher D. Manning (2017). «Deep Biaffine Attention for Neural Dependency Parsing.» In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=Hk95PK9le.

Dyer, Chris et al. (July 2015). «Transition-Based Dependency Parsing with Stack Long Short-Term Memory.» In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 334–343. DOI: 10.3115/v1/P15-1033. URL: https://www.aclweb.org/anthology/P15-1033.

Dyer, Chris et al. (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*. San Diego, California: Association for Computational Linguistics, pp. 199–209. DOI: 10.18653/v1/N16-1024. URL: http://www.aclweb.org/anthology/N16-1024.

Eisner, Jason M. (1996). «Three New Probabilistic Models for Dependency Parsing: An Exploration.» In: *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*. URL: https://www.aclweb.org/anthology/C96-1058.

Elman, Jeffrey L (1990). «Finding structure in time.» In: *Cognitive science* 14.2, pp. 179–211.

Falenska, Agnieszka, Anders Björkelund, and Jonas Kuhn (July 2020). «Integrating Graph-Based and Transition-Based Dependency Parsers in the Deep Contextualized Era.» In: *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*. Online: Association for Computational Linguistics, pp. 25–39. DOI: 10.18653/v1/2020.iwpt-1.4. URL: https://www.aclweb.org/anthology/2020.iwpt-1.4.

Falenska, Agnieszka and Jonas Kuhn (July 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*. Florence, Italy: Association for Computational Linguistics, pp. 117–128. DOI: 10.18653/v1/P19-1012. URL: https://www.aclweb.org/anthology/P19-1012.

Fernández-González, Daniel and Carlos Gómez-Rodríguez (July 2012). «Improving Transition-Based Dependency Parsing with Buffer Transitions.» In: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Jeju Island, Korea: Association for Computational Linguistics, pp. 308–319. URL: https://www.aclweb.org/anthology/D12-1029.

— (July 2017). «A Full Non-Monotonic Transition System for Unrestricted Non-Projective Parsing.» In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 288–298. DOI: 10.18653/v1/P17-1027. URL: https://www.aclweb.org/anthology/P17-1027.

Fernández-González, Daniel and Carlos Gómez-Rodríguez (June 2018). «Non-Projective Dependency Parsing with Non-Local Transitions.» In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 693–700. DOI: 10.18653/v1/N18-2109. URL: https://www.aclweb.org/anthology/N18-2109.

— (June 2019). «Left-to-Right Dependency Parsing with Pointer Networks.» In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 710–716. DOI: 10.18653/v1/N19-1076. URL: https://www.aclweb.org/anthology/N19-1076.

Fernández-González, Daniel and André F. T. Martins (July 2015). «Parsing as Reduction.» In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 1523–1533. DOI: 10.3115/v1/P15-1147. URL: https://www.aclweb.org/anthology/P15-1147.

Fernández-González, Daniel and Carlos Gómez-Rodríguez (2020). *Multitask Pointer Network for Multi-Representational Parsing*. arXiv: 2009.09730 [cs.CL].

Ferrer-i-Cancho, Ramon, Carlos Gómez-Rodríguez, and Juan Luis Esteban (2018). «Are crossing dependencies really scarce?» In: *Physica A: Statistical Mechanics and its Applications* 493, pp. 311–329. ISSN: 0378-4371. DOI: http://dx.doi.org/10.1016/j.physa.2017.10.048. URL: https://doi.org/10.1016/j.physa.2017.10.048.

Fraser, Norman (1989). «Parsing and dependency grammar.» In: *UCL Working Papers in Linguistics 1: University College London*. Pp. 296–319.

Ginter, Filip et al. (2017). *CoNLL 2017 Shared Task - Automatically Annotated Raw Texts and Word Embeddings*. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University. URL: http://hdl.handle.net/11234/1-1989.

Goldberg, Yoav (2017). *Neural Network Methods in Natural Language Processing*. Synthesis lectures on human language technologies. Morgan & Claypool Publishers.

— (2019). *Assessing BERT's Syntactic Abilities*. arXiv: 1901.05287 [cs.CL].

Goldberg, Yoav and Michael Elhadad (June 2010). «An Efficient Algorithm for Easy-First Non-Directional Dependency Parsing.» In: *Human Language Technologies: The 2010 Annual Conference of*

the North American Chapter of the Association for Computational Linguistics. Los Angeles, California: Association for Computational Linguistics, pp. 742–750. URL: https://www.aclweb.org/anthology/N10-1115.

Gómez-Rodríguez, Carlos (Dec. 2016). «Restricted Non-Projectivity: Coverage vs. Efficiency.» In: *Comput. Linguist.* 42.4, pp. 809–817. ISSN: 0891-2017. DOI: 10.1162/COLI_a_00267. URL: http://dx.doi.org/10.1162/COLI_a_00267.

Gómez-Rodríguez, Carlos (Sept. 2018). *Proyecto docente e investigador, plaza 18/037 de Profesor Titular de Universidad*.

Gómez-Rodríguez, Carlos, Iago Alonso-Alonso, and David Vilares (2019). «How important is syntactic parsing accuracy? An empirical evaluation on rule-based sentiment analysis.» In: *Artificial Intelligence Review* 52.3, pp. 2081–2097. ISSN: 1573-7462. DOI: 10.1007/s10462-017-9584-0. URL: https://doi.org/10.1007/s10462-017-9584-0.

Gómez-Rodríguez, Carlos and Joakim Nivre (July 2010). «A Transition-Based Parser for 2-Planar Dependency Structures.» In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Uppsala, Sweden: Association for Computational Linguistics, pp. 1492–1501. URL: https://www.aclweb.org/anthology/P10-1151.

— (2013). «Divisible Transition Systems and Multiplanar Dependency Parsing.» In: *Computational Linguistics* 39.4, pp. 799–845. DOI: 10.1162/COLI_a_00150. URL: https://www.aclweb.org/anthology/J13-4002.

Gómez-Rodríguez, Carlos, Michalina Strzyz, and David Vilares (Dec. 2020). «A Unifying Theory of Transition-based and Sequence Labeling Parsing.» In: *Proceedings of the 28th International Conference on Computational Linguistics*. Barcelona, Spain (Online): International Committee on Computational Linguistics, pp. 3776–3793. URL: https://www.aclweb.org/anthology/2020.coling-main.336.

Gómez-Rodríguez, Carlos and David Vilares (2018). «Constituent Parsing as Sequence Labeling.» In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, pp. 1314–1324. DOI: 10.18653/v1/D18-1162. URL: https://www.aclweb.org/anthology/D18-1162.

González-Garduño, Ana Valeria and Anders Søgaard (Sept. 2017). «Using Gaze to Predict Text Readability.» In: *Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 438–443. DOI: 10.18653/v1/W17-5050. URL: https://www.aclweb.org/anthology/W17-5050.

Goot, Rob van der et al. (Apr. 2021). «Massive Choice, Ample Tasks (MaChAmp): A Toolkit for Multi-task Learning in NLP.» In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*. Online: Association for Computational Linguistics, pp. 176–197. URL: https://www.aclweb.org/anthology/2021.eacl-demos.22.

Graves, Alex and Jürgen Schmidhuber (2005). «Framewise phoneme classification with bidirectional LSTM and other neural network architectures.» In: *Neural networks* 18.5-6, pp. 602–610.

Gómez-Rodríguez, Carlos (2017). «Towards fast natural language parsing: FASTPARSE ERC Starting Grant.» In: *Procesamiento del Lenguaje Natural* 59.0, pp. 121–124. ISSN: 1989-7553. URL: http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/view/5501.

Hewitt, John and Christopher D. Manning (June 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)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4129–4138. DOI: 10.18653/v1/N19-1419. URL: https://www.aclweb.org/anthology/N19-1419.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). «Long short-term memory.» In: *Neural computation* 9.8, pp. 1735–1780.

Hollenstein, Nora, Maria Barrett, and Lisa Beinborn (May 2020). «Towards Best Practices for Leveraging Human Language Processing Signals for Natural Language Processing.» English. In: *Proceedings of the Second Workshop on Linguistic and Neurocognitive Resources*. Marseille, France: European Language Resources Association, pp. 15–27. ISBN: 979-10-95546-52-8. URL: https://www.aclweb.org/anthology/2020.lincr-1.3.

Hollenstein, Nora and Ce Zhang (June 2019). «Entity Recognition at First Sight: Improving NER with Eye Movement Information.» In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 1–10. DOI: 10.18653/v1/N19-1001. URL: https://www.aclweb.org/anthology/N19-1001.

Hollenstein, Nora et al. (2019). «Advancing NLP with cognitive language processing signals.» In: *arXiv preprint arXiv:1904.02682*.

Honnibal, Matthew, Yoav Goldberg, and Mark Johnson (Aug. 2013). «A Non-Monotonic Arc-Eager Transition System for Dependency Parsing.» In: *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*. Sofia, Bulgaria: Association

for Computational Linguistics, pp. 163–172. URL: https://www.aclweb.org/anthology/W13-3518.

Honnibal, Matthew and Mark Johnson (Sept. 2015). «An Improved Non-monotonic Transition System for Dependency Parsing.» In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 1373–1378. DOI: 10.18653/v1/D15-1162. URL: https://www.aclweb.org/anthology/D15-1162.

Huang, Zhiheng, Wei Xu, and Kai Yu (2015). *Bidirectional LSTM-CRF Models for Sequence Tagging*. arXiv: 1508.01991 [cs.CL].

Jawahar, Ganesh, Benoît Sagot, and Djamé Seddah (July 2019). «What Does BERT Learn about the Structure of Language?» In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, pp. 3651–3657. DOI: 10.18653/v1/P19-1356. URL: https://www.aclweb.org/anthology/P19-1356.

Jiao, Xiaoqi et al. (Nov. 2020). «TinyBERT: Distilling BERT for Natural Language Understanding.» In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, pp. 4163–4174. DOI: 10.18653/v1/2020.findings-emnlp.372. URL: https://www.aclweb.org/anthology/2020.findings-emnlp.372.

Kahane, Sylvain and Nicolas Mazziotta (July 2015). «Syntactic Polygraphs. A Formalism Extending Both Constituency and Dependency.» In: *Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2015)*. Chicago, USA: Association for Computational Linguistics, pp. 152–164. DOI: 10.3115/v1/W15-2313. URL: https://www.aclweb.org/anthology/W15-2313.

Kankanampati, Yash et al. (Dec. 2020). «Multitask Easy-First Dependency Parsing: Exploiting Complementarities of Different Dependency Representations.» In: *Proceedings of the 28th International Conference on Computational Linguistics*. Barcelona, Spain (Online): International Committee on Computational Linguistics, pp. 2497–2508. DOI: 10.18653/v1/2020.coling-main.225. URL: https://www.aclweb.org/anthology/2020.coling-main.225.

Kennedy, Alan, Robin Hill, and Joël Pynte (2003). «The Dundee corpus.» In: *Proceedings of the 12th European conference on eye movement*.

Kiperwasser, Eliyahu and Miguel Ballesteros (2018). «Scheduled multitask learning: From syntax to translation.» In: *Transactions of the Association for Computational Linguistics* 6, pp. 225–240.

Kiperwasser, Eliyahu and Yoav Goldberg (2016). «Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations.» In: *Transactions of the Association for Computational Linguistics* 4, pp. 313–327. DOI: 10.1162/tacl_a_00101. URL: https://www.aclweb.org/anthology/Q16-1023.

Kitaev, Nikita, Steven Cao, and Dan Klein (July 2019). «Multilingual Constituency Parsing with Self-Attention and Pre-Training.» In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, pp. 3499–3505. DOI: 10.18653/v1/P19-1340. URL: https://www.aclweb.org/anthology/P19-1340.

Kitaev, Nikita and Dan Klein (July 2018). «Constituency Parsing with a Self-Attentive Encoder.» In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 2676–2686. DOI: 10.18653/v1/P18-1249. URL: https://www.aclweb.org/anthology/P18-1249.

Klein, Dan and Christopher D. Manning (2002). «Fast Exact Inference with a Factored Model for Natural Language Parsing.» In: *Proceedings of the 15th International Conference on Neural Information Processing Systems*. NIPS'02. Cambridge, MA, USA: MIT Press, pp. 3–10. URL: http://dl.acm.org/citation.cfm?id=2968618.2968619.

Klerke, Sigrid, Yoav Goldberg, and Anders Søgaard (June 2016). «Improving sentence compression by learning to predict gaze.» In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, pp. 1528–1533. DOI: 10.18653/v1/N16-1179. URL: https://www.aclweb.org/anthology/N16-1179.

Klerke, Sigrid and Barbara Plank (Nov. 2019). «At a Glance: The Impact of Gaze Aggregation Views on Syntactic Tagging.» In: *Proceedings of the Beyond Vision and LANguage: inTEgrating Real-world kNowledge (LANTERN)*. Hong Kong, China: Association for Computational Linguistics, pp. 51–61. DOI: 10.18653/v1/D19-6408. URL: https://www.aclweb.org/anthology/D19-6408.

Kondratyuk, Dan and Milan Straka (Nov. 2019). «75 Languages, 1 Model: Parsing Universal Dependencies Universally.» In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, pp. 2779–2795. DOI: 10.18653/v1/D19-1279. URL: https://www.aclweb.org/anthology/D19-1279.

Koo, Terry and Michael Collins (July 2010). «Efficient Third-Order Dependency Parsers.» In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Uppsala, Sweden: Association for Computational Linguistics, pp. 1–11. URL: http://www.aclweb.org/anthology/P10-1001.

Kübler, S., R. McDonald, and J. Nivre (2009). *Dependency Parsing*. Synthesis lectures on human language technologies. Morgan & Claypool. ISBN: 9781598295962.

Kuhlmann, Marco, Carlos Gómez-Rodríguez, and Giorgio Satta (June 2011). «Dynamic Programming Algorithms for Transition-Based Dependency Parsers.» In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, pp. 673–682. URL: https://www.aclweb.org/anthology/P11-1068.

Kulmizev, Artur et al. (Nov. 2019). «Deep Contextualized Word Embeddings in Transition-Based and Graph-Based Dependency Parsing - A Tale of Two Parsers Revisited.» In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, pp. 2755–2768. DOI: 10.18653/v1/D19-1277. URL: https://www.aclweb.org/anthology/D19-1277.

Kuncoro, Adhiguna et al. (Nov. 2016). «Distilling an Ensemble of Greedy Dependency Parsers into One MST Parser.» In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, pp. 1744–1753. DOI: 10.18653/v1/D16-1180. URL: https://www.aclweb.org/anthology/D16-1180.

Lacroix, Ophélie (Aug. 2019). «Dependency Parsing as Sequence Labeling with Head-Based Encoding and Multi-Task Learning.» In: *Proceedings of the Fifth International Conference on Dependency Linguistics (Depling, SyntaxFest 2019)*. Paris, France: Association for Computational Linguistics, pp. 136–143. DOI: 10.18653/v1/W19-7716. URL: https://aclanthology.org/W19-7716.

Lafferty, John D., Andrew McCallum, and Fernando C. N. Pereira (2001). «Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data.» In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 282–289. ISBN: 1558607781.

Lample, Guillaume et al. (June 2016). «Neural Architectures for Named Entity Recognition.» In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, pp. 260–270. DOI: 10.18653/v1/N16-1030. URL: https://www.aclweb.org/anthology/N16-1030.

Levy, Omer and Yoav Goldberg (June 2014). «Dependency-Based Word Embeddings.» In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Pa-*

pers). Baltimore, Maryland: Association for Computational Linguistics, pp. 302–308. DOI: 10.3115/v1/P14-2050. URL: https://www.aclweb.org/anthology/P14-2050.

Lhoneux, Miryam de, Sara Stymne, and Joakim Nivre (Sept. 2017a). «Arc-Hybrid Non-Projective Dependency Parsing with a Static-Dynamic Oracle.» In: *Proceedings of the 15th International Conference on Parsing Technologies*. Pisa, Italy: Association for Computational Linguistics, pp. 99–104. URL: https://www.aclweb.org/anthology/W17-6314.

— (2017b). «Old School vs. New School: Comparing Transition-Based Parsers with and without Neural Network Enhancement.» In: *Proceedings of the 15th International Workshop on Treebanks and Linguistic Theories (TLT15), Bloomington, IN, USA, January 20-21, 2017*. Ed. by Markus Dickinson et al. Vol. 1779. CEUR Workshop Proceedings. CEUR-WS.org, pp. 99–110. URL: http://ceur-ws.org/Vol-1779/08delhoneux.pdf.

Lhoneux, Miryam de et al. (Aug. 2017). «From Raw Text to Universal Dependencies - Look, No Tags!» In: *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Vancouver, Canada: Association for Computational Linguistics, pp. 207–217. DOI: 10.18653/v1/K17-3022. URL: https://www.aclweb.org/anthology/K17-3022.

Li, Zuchao et al. (Aug. 2018). «Seq2seq Dependency Parsing.» In: *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, pp. 3203–3214. URL: https://www.aclweb.org/anthology/C18-1271.

Liebling, Daniel J and Sören Preibusch (2014). «Privacy considerations for a pervasive eye tracking world.» In: *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, pp. 1169–1177.

Ling, Wang et al. (2015). «Two/Too Simple Adaptations of Word2Vec for Syntax Problems.» In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Denver, Colorado: Association for Computational Linguistics, pp. 1299–1304. DOI: 10.3115/v1/N15-1142. URL: https://www.aclweb.org/anthology/N15-1142.

Liu, Zihan, Genta Indra Winata, and Pascale Fung (July 2020). «Zero-Resource Cross-Domain Named Entity Recognition.» In: *Proceedings of the 5th Workshop on Representation Learning for NLP*. Online: Association for Computational Linguistics, pp. 1–6. DOI: 10.18653/v1/2020.repl4nlp-1.1. URL: https://www.aclweb.org/anthology/2020.repl4nlp-1.1.

Lopopolo, Alessandro et al. (June 2019). «Dependency Parsing with your Eyes: Dependency Structure Predicts Eye Regressions During

Reading.» In: *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 77–85. DOI: 10.18653/v1/W19-2909. URL: https://www.aclweb.org/anthology/W19-2909.

Ma, Xuezhe and Eduard Hovy (Aug. 2016). «End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF.» In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 1064–1074. DOI: 10.18653/v1/P16-1101. URL: https://www.aclweb.org/anthology/P16-1101.

Ma, Xuezhe et al. (July 2018). «Stack-Pointer Networks for Dependency Parsing.» In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 1403–1414. DOI: 10.18653/v1/P18-1130. URL: https://www.aclweb.org/anthology/P18-1130.

Marcus, Mitchell P, Mary Ann Marcinkiewicz, and Beatrice Santorini (1993). «Building a large annotated corpus of English: The Penn Treebank.» In: *Computational linguistics* 19.2, pp. 313–330.

Marneffe, Marie-Catherine de, Bill MacCartney, and Christopher D. Manning (May 2006). «Generating Typed Dependency Parses from Phrase Structure Parses.» In: *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*. Genoa, Italy: European Language Resources Association (ELRA). URL: http://www.lrec-conf.org/proceedings/lrec2006/pdf/440_pdf.pdf.

Martínez Alonso, Héctor and Barbara Plank (Apr. 2017). «When is multitask learning effective? Semantic sequence prediction under varying data conditions.» In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Valencia, Spain: Association for Computational Linguistics, pp. 44–53. URL: https://www.aclweb.org/anthology/E17-1005.

Martins, André, Miguel Almeida, and Noah A. Smith (Aug. 2013). «Turning on the Turbo: Fast Third-Order Non-Projective Turbo Parsers.» In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 617–622. URL: https://www.aclweb.org/anthology/P13-2109.

McDonald, Ryan, Koby Crammer, and Fernando Pereira (June 2005). «Online Large-Margin Training of Dependency Parsers.» In: *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*. Ann Arbor, Michigan: Association for Computational Linguistics, pp. 91–98. DOI: 10.3115/1219840.1219852. URL: https://www.aclweb.org/anthology/P05-1012.

McDonald, Ryan and Joakim Nivre (2011). «Analyzing and Integrating Dependency Parsers.» In: *Computational Linguistics* 37.1, pp. 197–230. DOI: 10.1162/coli_a_00039. URL: https://www.aclweb.org/anthology/J11-1007.

McDonald, Ryan and Giorgio Satta (June 2007). «On the Complexity of Non-Projective Data-Driven Dependency Parsing.» In: *Proceedings of the Tenth International Conference on Parsing Technologies*. Prague, Czech Republic: Association for Computational Linguistics, pp. 121–132. URL: https://www.aclweb.org/anthology/W07-2216.

Mel'cuk, Igor Aleksandrovic (1988). *Dependency syntax: theory and practice*. SUNY press.

Mikolov, Tomas et al. (2013). *Efficient Estimation of Word Representations in Vector Space*. arXiv: 1301.3781 [cs.CL].

Mishra, Abhijit et al. (Aug. 2016). «Leveraging Cognitive Features for Sentiment Analysis.» In: *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*. Berlin, Germany: Association for Computational Linguistics, pp. 156–166. DOI: 10.18653/v1/K16-1016. URL: https://www.aclweb.org/anthology/K16-1016.

Mrini, Khalil et al. (2019). *Rethinking Self-Attention: Towards Interpretability in Neural Parsing*. arXiv: 1911.03875 [cs.CL].

Mrini, Khalil et al. (Nov. 2020). «Rethinking Self-Attention: Towards Interpretability in Neural Parsing.» In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, pp. 731–742. DOI: 10.18653/v1/2020.findings-emnlp.65. URL: https://www.aclweb.org/anthology/2020.findings-emnlp.65.

Nivre, Joakim (Apr. 2003). «An Efficient Algorithm for Projective Dependency Parsing.» In: *Proceedings of the Eighth International Conference on Parsing Technologies*. Nancy, France, pp. 149–160. URL: https://www.aclweb.org/anthology/W03-3017.

— (July 2004). «Incrementality in Deterministic Dependency Parsing.» In: *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*. Barcelona, Spain: Association for Computational Linguistics, pp. 50–57. URL: https://www.aclweb.org/anthology/W04-0308.

— (2008). «Algorithms for Deterministic Incremental Dependency Parsing.» In: *Computational Linguistics* 34.4, pp. 513–553. DOI: 10.1162/coli.07-056-R1-07-027. URL: https://www.aclweb.org/anthology/J08-4003.

— (Aug. 2009). «Non-Projective Dependency Parsing in Expected Linear Time.» In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*. Suntec, Singa-

pore: Association for Computational Linguistics, pp. 351–359. URL: https://www.aclweb.org/anthology/P09-1040.

Nivre, Joakim and Daniel Fernández-González (June 2014). «Squibs: Arc-Eager Parsing with the Tree Constraint.» In: *Computational Linguistics* 40.2, pp. 259–267. DOI: 10.1162/COLI_a_00185. URL: https://www.aclweb.org/anthology/J14-2002.

Nivre, Joakim, Johan Hall, and Jens Nilsson (May 2006). «MaltParser: A Data-Driven Parser-Generator for Dependency Parsing.» In: URL: http://www.lrec-conf.org/proceedings/lrec2006/pdf/162_pdf.pdf.

Nivre, Joakim and Ryan McDonald (June 2008). «Integrating Graph-Based and Transition-Based Dependency Parsers.» In: *Proceedings of ACL-08: HLT*. Columbus, Ohio: Association for Computational Linguistics, pp. 950–958. URL: https://www.aclweb.org/anthology/P08-1108.

Nivre, Joakim et al. (2007). «MaltParser: A language-independent system for data-driven dependency parsing.» In: *Natural Language Engineering* 13.2, p. 95.

Nivre, Joakim et al. (May 2016). «Universal Dependencies v1: A Multilingual Treebank Collection.» In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*. Portorož, Slovenia: European Language Resources Association (ELRA), pp. 1659–1666. URL: https://www.aclweb.org/anthology/L16-1262.

Nivre, Joakim et al. (2018). *Universal Dependencies 2.2*. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University. URL: http://hdl.handle.net/11234/1-2837.

— (2019). *Universal Dependencies 2.4*. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University. URL: http://hdl.handle.net/11234/1-2988.

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2013). «On the difficulty of training recurrent neural networks.» In: *International conference on machine learning*. PMLR, pp. 1310–1318.

Pei, Wenzhe, Tao Ge, and Baobao Chang (July 2015). «An Effective Neural Network Model for Graph-based Dependency Parsing.» In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 313–322. DOI: 10.3115/v1/P15-1031. URL: https://www.aclweb.org/anthology/P15-1031.

Pennington, Jeffrey, Richard Socher, and Christopher Manning (Oct. 2014). «GloVe: Global Vectors for Word Representation.» In: *Proceedings of the 2014 Conference on Empirical Methods in Natural*

Language Processing (EMNLP). Doha, Qatar: Association for Computational Linguistics, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://www.aclweb.org/anthology/D14-1162.

Peters, Matthew et al. (June 2018). «Deep Contextualized Word Representations.» In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 2227–2237. DOI: 10.18653/v1/N18-1202. URL: https://www.aclweb.org/anthology/N18-1202.

Pitler, Emily and Ryan McDonald (May 2015). «A Linear-Time Transition System for Crossing Interval Trees.» In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Denver, Colorado: Association for Computational Linguistics, pp. 662–671. DOI: 10.3115/v1/N15-1068. URL: https://www.aclweb.org/anthology/N15-1068.

Plank, Barbara, Anders Søgaard, and Yoav Goldberg (Aug. 2016). «Multilingual Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Models and Auxiliary Loss.» In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 412–418. DOI: 10.18653/v1/P16-2067. URL: https://www.aclweb.org/anthology/P16-2067.

Qi, Peng and Christopher D. Manning (July 2017). «Arc-swift: A Novel Transition System for Dependency Parsing.» In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 110–117. DOI: 10.18653/v1/P17-2018. URL: https://www.aclweb.org/anthology/P17-2018.

Rayner, Keith (1998). «Eye movements in reading and information processing: 20 years of research.» In: *Psychological bulletin* 124.3, p. 372.

— (2009). «The 35th Sir Frederick Bartlett Lecture: Eye movements and attention in reading, scene perception, and visual search.» In: *Quarterly journal of experimental psychology* 62.8, pp. 1457–1506.

Ren, Xiaona, Xiao Chen, and Chunyu Kit (2013). «Combine Constituent and Dependency Parsing via Reranking.» In: *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*. IJCAI '13. Beijing, China: AAAI Press, pp. 2155–2161. ISBN: 978-1-57735-633-2. URL: http://dl.acm.org/citation.cfm?id=2540128.2540438.

Ruder, Sebastian (2017). «An Overview of Multi-Task Learning in Deep Neural Networks.» In: *CoRR* abs/1706.05098. arXiv: 1706.05098. URL: http://arxiv.org/abs/1706.05098.

Salama, Amr Rekaby and Wolfgang Menzel (2016). «Multimodal graph-based dependency parsing of natural language.» In: *International Conference on Advanced Intelligent Systems and Informatics*. Springer, pp. 22–31.

Sartorio, Francesco, Giorgio Satta, and Joakim Nivre (Aug. 2013). «A Transition-Based Dependency Parser Using a Dynamic Parsing Strategy.» In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 135–144. URL: https://www.aclweb.org/anthology/P13-1014.

Schröder, Fynn and Chris Biemann (July 2020). «Estimating the influence of auxiliary tasks for multi-task learning of sequence tagging tasks.» In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, pp. 2971–2985. DOI: 10.18653/v1/2020.acl-main.268. URL: https://www.aclweb.org/anthology/2020.acl-main.268.

Schwartz, Roy et al. (2019). «Green AI.» In: *CoRR* abs/1907.10597. arXiv: 1907.10597. URL: http://arxiv.org/abs/1907.10597.

Seddah, Djamé et al. (Oct. 2013). «Overview of the SPMRL 2013 Shared Task: A Cross-Framework Evaluation of Parsing Morphologically Rich Languages.» In: *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*. Seattle, Washington, USA: Association for Computational Linguistics, pp. 146–182. URL: https://aclanthology.org/W13-4917.

Shi, Tianze, Liang Huang, and Lillian Lee (Sept. 2017). «Fast(er) Exact Decoding and Global Training for Transition-Based Dependency Parsing via a Minimal Feature Set.» In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 12–23. DOI: 10.18653/v1/D17-1002. URL: https://www.aclweb.org/anthology/D17-1002.

Smith, Aaron et al. (Oct. 2018). «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*. Brussels, Belgium: Association for Computational Linguistics, pp. 113–123. DOI: 10.18653/v1/K18-2011. URL: https://www.aclweb.org/anthology/K18-2011.

Søgaard, Anders (Aug. 2016). «Evaluating word embeddings with fMRI and eye-tracking.» In: *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*. Berlin, Germany: Association for Computational Linguistics, pp. 116–121. DOI: 10.18653/v1/W16-2521. URL: https://www.aclweb.org/anthology/W16-2521.

Søgaard, Anders and Yoav Goldberg (Aug. 2016). «Deep multi-task learning with low level tasks supervised at lower layers.» In: *Pro-*

ceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). Berlin, Germany: Association for Computational Linguistics, pp. 231–235. DOI: 10.18653/v1/P16-2038. URL: https://www.aclweb.org/anthology/P16-2038.

Song, Kaiqiang, Lin Zhao, and Fei Liu (Aug. 2018). «Structure-Infused Copy Mechanisms for Abstractive Summarization.» In: *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, pp. 1717–1729. URL: https://www.aclweb.org/anthology/C18-1146.

Spoustová, Drahomíra and Miroslav Spousta (2010). «Dependency Parsing as a Sequence Labeling Task.» In: *The Prague Bulletin of Mathematical Linguistics* 94.1, pp. 7–14. URL: https://content.sciendo.com/view/journals/pralin/94/1/article-p7.xml.

Straka, Milan and Jana Straková (Aug. 2017). «Tokenizing, POS Tagging, Lemmatizing and Parsing UD 2.0 with UDPipe.» In: *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Vancouver, Canada: Association for Computational Linguistics, pp. 88–99. DOI: 10.18653/v1/K17-3009. URL: https://www.aclweb.org/anthology/K17-3009.

Strubell, Emma, Ananya Ganesh, and Andrew McCallum (July 2019). «Energy and Policy Considerations for Deep Learning in NLP.» In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, pp. 3645–3650. DOI: 10.18653/v1/P19-1355. URL: https://www.aclweb.org/anthology/P19-1355.

Strubell, Emma and Andrew McCallum (Sept. 2017). «Dependency Parsing with Dilated Iterated Graph CNNs.» In: *Proceedings of the 2nd Workshop on Structured Prediction for Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 1–6. DOI: 10.18653/v1/W17-4301. URL: https://www.aclweb.org/anthology/W17-4301.

Strubell, Emma et al. (Oct. 2018). «Linguistically-Informed Self-Attention for Semantic Role Labeling.» In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, pp. 5027–5038. DOI: 10.18653/v1/D18-1548. URL: https://www.aclweb.org/anthology/D18-1548.

Strzyz, Michalina and Carlos Gómez-Rodríguez (2019). «Speeding up Natural Language Parsing by Reusing Partial Results.» In: *arXiv* 1904.03417 [cs.CL]. URL: https://arxiv.org/abs/1904.03417.

Strzyz, Michalina, David Vilares, and Carlos Gómez-Rodríguez (July 2019a). «Sequence Labeling Parsing by Learning across Representations.» In: *Proceedings of the 57th Annual Meeting of the Associa-*

tion for Computational Linguistics. Florence, Italy: Association for Computational Linguistics, pp. 5350–5357. DOI: 10.18653/v1/P19-1531. URL: https://www.aclweb.org/anthology/P19-1531.

— (Nov. 2019b). «Towards Making a Dependency Parser See.» In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China: Association for Computational Linguistics, pp. 1500–1506. DOI: 10.18653/v1/D19-1160. URL: https://www.aclweb.org/anthology/D19-1160.

— (June 2019c). «Viable Dependency Parsing as Sequence Labeling.» In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 717–723. DOI: 10.18653/v1/N19-1077. URL: https://www.aclweb.org/anthology/N19-1077.

— (Dec. 2020). «Bracketing Encodings for 2-Planar Dependency Parsing.» In: *Proceedings of the 28th International Conference on Computational Linguistics*. Barcelona, Spain (Online): International Committee on Computational Linguistics, pp. 2472–2484. URL: https://www.aclweb.org/anthology/2020.coling-main.223.

Tenney, Ian et al. (2019). *What do you learn from context? Probing for sentence structure in contextualized word representations*. arXiv: 1905.06316 [cs.CL].

Tesnière, L. (1959). *Élements de syntaxe structurale*. C. Klincksieck.

Titov, Ivan and James Henderson (June 2007). «A Latent Variable Model for Generative Dependency Parsing.» In: *Proceedings of the Tenth International Conference on Parsing Technologies*. Prague, Czech Republic: Association for Computational Linguistics, pp. 144–155. URL: https://www.aclweb.org/anthology/W07-2218.

Toneva, Mariya and Leila Wehbe (2019). «Interpreting and improving natural-language processing (in machines) with natural language-processing (in the brain).» In: *Advances in Neural Information Processing Systems*, pp. 14954–14964.

Toutanova, Kristina et al. (2003). «Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network.» In: *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 252–259. URL: https://www.aclweb.org/anthology/N03-1033.

Vaswani, Ashish et al. (2017). «Attention is All you Need.» In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., pp. 5998–6008. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Vieira, Tim and Jason Eisner (2017). «Learning to Prune: Exploring the Frontier of Fast and Accurate Parsing.» In: *Transactions of the Association for Computational Linguistics* 5, pp. 263–278. DOI: 10.1162/tacl_a_00060. URL: https://www.aclweb.org/anthology/Q17-1019.

Vilares, David, Mostafa Abdou, and Anders Søgaard (June 2019). «Better, Faster, Stronger Sequence Tagging Constituent Parsers.» In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 3372–3383. DOI: 10.18653/v1/N19-1341. URL: https://www.aclweb.org/anthology/N19-1341.

Vilares, David and Carlos Gómez-Rodríguez (Nov. 2018). «Transition-based Parsing with Lighter Feed-Forward Networks.» In: *Proceedings of the Second Workshop on Universal Dependencies (UDW 2018)*. Brussels, Belgium: Association for Computational Linguistics, pp. 162–172. DOI: 10.18653/v1/W18-6019. URL: https://aclanthology.org/W18-6019.

— (Nov. 2020). «Discontinuous Constituent Parsing as Sequence Labeling.» In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, pp. 2771–2785. DOI: 10.18653/v1/2020.emnlp-main.221. URL: https://aclanthology.org/2020.emnlp-main.221.

Vilares, David et al. (2020). «Parsing as Pretraining.» In: *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, pp. 9114–9121. DOI: https://doi.org/10.1609/aaai.v34i05.6446. URL: https://aaai.org/ojs/index.php/AAAI/article/view/6446.

Vinyals, Oriol, Meire Fortunato, and Navdeep Jaitly (2015). «Pointer Networks.» In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc., pp. 2692–2700. URL: https://proceedings.neurips.cc/paper/2015/file/29921001f2f04bd3baee84a12e98098f-Paper.pdf.

Vinyals, Oriol et al. (2015). «Grammar as a foreign language.» In: *Advances in neural information processing systems* 28, pp. 2773–2781.

Virtanen, Antti et al. (2019). «Multilingual is not enough: BERT for Finnish.» In: *arXiv preprint arXiv:1912.07076*.

Wang, Hanrui et al. (July 2020). «HAT: Hardware-Aware Transformers for Efficient Natural Language Processing.» In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, pp. 7675–7688. DOI: 10.18653/v1/2020.acl-main.686. URL: https://www.aclweb.org/anthology/2020.acl-main.686.

Wang, Wenhui and Baobao Chang (Aug. 2016). «Graph-based Dependency Parsing with Bidirectional LSTM.» In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 2306–2315. DOI: 10.18653/v1/P16-1218. URL: https://www.aclweb.org/anthology/P16-1218.

Wu, Shijie and Mark Dredze (July 2020). «Are All Languages Created Equal in Multilingual BERT?» In: *Proceedings of the 5th Workshop on Representation Learning for NLP*. Online: Association for Computational Linguistics, pp. 120–130. DOI: 10.18653/v1/2020.repl4nlp-1.16. URL: https://aclanthology.org/2020.repl4nlp-1.16.

Wu, Yonghui et al. (2016). «Google's neural machine translation system: Bridging the gap between human and machine translation.» In: *arXiv preprint arXiv:1609.08144*. URL: https://arxiv.org/abs/1609.08144.

Yamada, Hiroyasu and Yuji Matsumoto (Apr. 2003). «Statistical Dependency Analysis with Support Vector Machines.» In: *Proceedings of the Eighth International Conference on Parsing Technologies*. Nancy, France, pp. 195–206. URL: https://www.aclweb.org/anthology/W03-3023.

Yang, Jie and Yue Zhang (July 2018). «NCRF++: An Open-source Neural Sequence Labeling Toolkit.» In: pp. 74–79. DOI: 10.18653/v1/P18-4013. URL: https://www.aclweb.org/anthology/P18-4013.

Yang, Zhilin, Ruslan Salakhutdinov, and William W. Cohen (2016). «Multi-Task Cross-Lingual Sequence Tagging from Scratch.» In: *CoRR* abs/1603.06270. arXiv: 1603.06270. URL: http://arxiv.org/abs/1603.06270.

Yang, Zhilin, Ruslan Salakhutdinov, and William W Cohen (2017). «Transfer learning for sequence tagging with hierarchical recurrent networks.» In: *arXiv preprint arXiv:1703.06345*.

Yang, Zhilin et al. (2019). «XLNet: Generalized Autoregressive Pretraining for Language Understanding.» In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., pp. 5753–5763. URL: https://proceedings.neurips.cc/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf.

Yli-Jyrä, Anssi Mikael (2003). «Multiplanarity – a Model for Dependency Structures in Treebanks.» In: *TLT 2003. Proceedings of the Second Workshop on Treebanks and Linguistic Theories*. Ed. by Joakim Nivre and Erhard Hinrichs. Vol. 9. Mathematical Modelling in Physics, Engineering and Cognitive Sciences. Växjö, Sweden: Växjö University Press, pp. 189–200.

Yli-Jyrä, Anssi (2012). «On Dependency Analysis via Contractions and Weighted FSTs.» In: *Shall We Play the Festschrift Game? Essays on the Occasion of Lauri Carlson's 60th Birthday*. Ed. by Diana Santos, Krister Lindén, and Wanjiku Ng'ang'a. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 133–158. ISBN: 978-3-642-30773-7. DOI: 10.1007/978-3-642-30773-7_10. URL: https://doi.org/10.1007/978-3-642-30773-7_10.

Yli-Jyrä, Anssi and Carlos Gómez-Rodríguez (July 2017). «Generic Axiomatization of Families of Noncrossing Graphs in Dependency Parsing.» In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 1745–1755. ISBN: 978-1-945626-75-3. URL: http://aclweb.org/anthology/P17-1160.

Zeman, Daniel et al. (2021). *Universal Dependencies 2.8.1*. URL: http://hdl.handle.net/11234/1-3687.

Zhang, Bo et al. (July 2020). «Syntax-Aware Opinion Role Labeling with Dependency Graph Convolutional Networks.» In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, pp. 3249–3258. DOI: 10.18653/v1/2020.acl-main.297. URL: https://www.aclweb.org/anthology/2020.acl-main.297.

Zhang, Meishan et al. (June 2019). «Syntax-Enhanced Neural Machine Translation with Syntax-Aware 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)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 1151–1161. DOI: 10.18653/v1/N19-1118. URL: https://www.aclweb.org/anthology/N19-1118.

Zhang, X. et al. (2018). «Converting Your Thoughts to Texts: Enabling Brain Typing via Deep Feature Learning of EEG Signals.» In: *2018 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 1–10.

Zhang, Xingxing, Jianpeng Cheng, and Mirella Lapata (Apr. 2017). «Dependency Parsing as Head Selection.» In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Valencia, Spain: Association for Computational Linguistics, pp. 665–676. URL: https://www.aclweb.org/anthology/E17-1063.

Zhang, Yu, Zhenghua Li, and Min Zhang (July 2020). «Efficient Second-Order TreeCRF for Neural Dependency Parsing.» In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, pp. 3295–3305. DOI: 10.18653/v1/2020.acl-main.302. URL: https://www.aclweb.org/anthology/2020.acl-main.302.

Zhang, Yu, Houquan Zhou, and Zhenghua Li (2020). «Fast and Accurate Neural CRF Constituency Parsing.» In: *Proceedings of IJCAI*, pp. 4046–4053. DOI: 10.24963/ijcai.2020/560. URL: https://doi.org/10.24963/ijcai.2020/560.

Zhang, Yue and Stephen Clark (Oct. 2008). «A Tale of Two Parsers: Investigating and Combining Graph-based and Transition-based Dependency Parsing.» In: *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*. Honolulu, Hawaii: Association for Computational Linguistics, pp. 562–571. URL: https://www.aclweb.org/anthology/D08-1059.

Zhang, Yue and Joakim Nivre (June 2011). «Transition-based Dependency Parsing with Rich Non-local Features.» In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, pp. 188–193. URL: https://www.aclweb.org/anthology/P11-2033.

Zhou, Ganbin et al. (2018). «Tree-Structured Neural Machine for Linguistics-Aware Sentence Generation.» In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. Ed. by Sheila A. McIlraith and Kilian Q. Weinberger. AAAI Press, pp. 5722–5729. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16567.

Zhou, Junru, Zuchao Li, and Hai Zhao (Nov. 2020). «Parsing All: Syntax and Semantics, Dependencies and Spans.» In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, pp. 4438–4449. DOI: 10.18653/v1/2020.findings-emnlp.398. URL: https://www.aclweb.org/anthology/2020.findings-emnlp.398.

Zhou, Junru and Hai Zhao (July 2019). «Head-Driven Phrase Structure Grammar Parsing on Penn Treebank.» In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, pp. 2396–2408. DOI: 10.18653/v1/P19-1230. URL: https://www.aclweb.org/anthology/P19-1230.

Zhu, Muhua et al. (Aug. 2013). «Fast and Accurate Shift-Reduce Constituent Parsing.» In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 434–443. URL: https://www.aclweb.org/anthology/P13-1043.

Model details

A.1 Model parameters

A.1.1 NCRF++: Setup 1

Hyperparameters	Value
Loss	cross-entropy
Batch size	training:8 test:128
Optimizer	Stochastic Gradient Descent (SGD)
Epochs	100
Initial learning rate	0.02
Learning rate decay	0.05
Momentum	0.9
Dropout	0.5
Word embeddings	100
Char embeddings	30
PoS tag embeddings	25
BiLSTM size	800 (400:LSTM$^{\rightarrow}$ and 400:LSTM$^{\leftarrow}$)
Character hidden vector	50
Bi-LSTM layer	2

Table 41: Hyperparameters for the BiLSTM model in setup 1.

We keep a model that obtains the highest LAS on the development set during training with exception of the experiments from Section 3.3.1, Section 4.3.1.1 and Section 4.4.1.4, in which the models are chosen based on the highest UAS. For experiments with the Penn Treebank (PTB), we use the word embeddings proposed by Li et al., 2018, while for UD languages we use the word embeddings from Ginter et al., 2017 for available languages.

A.1.2 NCRF++: Setup 2

Hyperparameters	Value
Loss	cross-entropy
Batch size	training:8 test:128
Optimizer	Stochastic Gradient Descent (SGD)
Epochs	150
Initial learning rate	0.02
Learning rate decay	0.05
Momentum	0.9
Dropout	0.5
Word embeddings	100
Char embeddings	30
Self-defined features emb.	20 [1]
BiLSTM size	800 (400:LSTM$^{\rightarrow}$ and 400:LSTM$^{\leftarrow}$)
Character hidden vector	50
Bi-LSTM layer	2

Table 42: Hyperparameters for BiLSTM model in setup 2.

MTL model	Weighting factor
2-task D	1
3-task C	1
D with auxiliary task C	D: 1 and C: 0.2
C with auxiliary task D	C: 1 and D: 0.1
Multi-task C and D	1

Table 43: Weighting factors used for learning representations from dependency parsing (denoted with D) and constituency parsing (C).

We keep a dependency parser model with the highest LAS score obtained on the dev set during training, while for constituency parsing we keep a model with the highest F1 score. In case of using both paradigms as main tasks in the MTL setup, a model is chosen based on the highest harmonic mean among LAS and F1 scores. In this setup, we only use pre-trained word embeddings in models trained on the PTB treebank provided by Li et al., 2018, but not for other languages.

1 In case of the models trained on the PTB treebank, we use PoS tags features with embeddings size of 25.

A.1.3 BERT

Hyperparameters	Value
Loss	cross-entropy
Batch size	8
Epochs	45
Learning rate	10^{-5}
Max.seq.length	400 (510 Russian)

Table 44: Hyperparameters for BERT.

The best model is chosen based on the highest LAS score obtained on the dev set during training.

B Further analyses

B.1 Statistics on highly non-projective treebanks

We provide statistics relevant for the bracketing-based family of encodings described in Chapter 4 which are tested on highly non-projective UDv2.4 treebanks. Table 45 provides the size of treebanks sorted by the decreasing percentage of non-projective sentences they contain. It is worth mentioning that we excluded some treebanks due to lack of pre-trained UDPipe models or a development set.

Language	Train	Dev	Test
Ancient $\text{Greek}_{\text{Perseus}}$	11476 (*62.77%*)	1137 (*74.41%*)	1306 (*64.40%*)
$\text{Basque}_{\text{BDT}}$	5396 (*33.52%*)	1798 (*33.48%*)	1799 (*31.80%*)
$\text{Hungarian}_{\text{Szeged}}$	910 (*25.71%*)	441 (*33.56%*)	449 (*23.61%*)
$\text{Portuguese}_{\text{Bosque}}$	8328 (*23.60%*)	560 (*19.46%*)	477 (*22.85%*)
$\text{Urdu}_{\text{UDTB}}$	4043 (*23.00%*)	552 (*23.01%*)	535 (*18.88%*)
$\text{Afrikaans}_{\text{AfriBooms}}$	1315 (*22.21%*)	194 (*20.10%*)	425 (*23.76%*)
$\text{Korean}_{\text{Kaist}}$	23010 (*21.92%*)	2066 (*22.12%*)	2287 (*19.15%*)
$\text{Danish}_{\text{DDT}}$	4383 (*21.83%*)	564 (*21.81%*)	565 (*18.58%*)
$\text{Gothic}_{\text{PROIEL}}$	3387 (*16.77%*)	985 (*19.09%*)	1029 (*18.76%*)
$\text{Lithuanian}_{\text{HSE}}$	153 (*16.34%*)	55 (*16.36%*)	55 (*21.82%*)

Table 45: Total number of sentences in the training, development and test splits for the selected highly non-projective UD treebanks. We provide the percentage of non-projective sentences in parentheses.

B.2 Impact of the training data size

We examine the effect of training data size on the performance of our sequence labeling parsers compared to the transition-based BIST parser (Kiperwasser and Goldberg, 2016). Figure 32 illustrates the UAS and LAS scores obtained by each parser with respect to the percentage of PTB size used for training. The sequence labeling model relies on the relative PoS-based encoding with the hyperparameters from Section A.1.1.

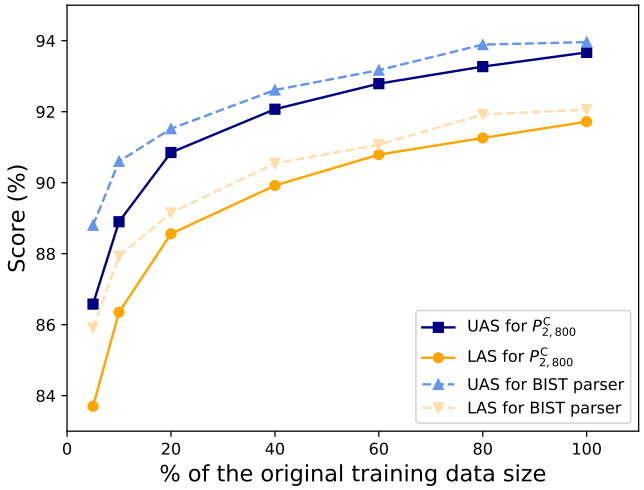


Figure 32: Impact of the PTB data size available for PoS-based models and BIST parser during training on the UAS and LAS results on the test set.

In general, the results suggest that our model requires more training data than the BIST parser in order to obtain similar accuracy. This is most prominent when there is only a small amount of labeled data available (for instance, when using only 10% of PTB training data). However, this discrepancy in accuracy diminishes with the increase of training data.

C

Resumen largo en español

En esta tesis presentamos nuevos métodos para reformular el análisis sintáctico de dependencias como una tarea de etiquetado secuencial que puede servir como una alternativa viable a los enfoques basados en transiciones y grafos. Para ello, proponemos un conjunto de métodos de codificación que hacen corresponder a cada árbol de dependencias una secuencia etiquetas discretas, de tal manera que cada palabra de la oración se asocia con una, y solo una etiqueta. Trabajo previo en este sentido (Spoustová y Spousta, 2010) ha sugerido que una formulación del análisis sintáctico como etiquetado secuencial era poco práctica. Sin embargo, en esta tesis, revisamos y ampliamos estos intentos previos aprovechando recientes avances en modelos de redes neuronales que pueden ser aplicados al procesamiento del lenguaje natural. En relación a esto, Gómez-Rodríguez y Vilares (2018) han demostrado recientemente que este tipo de enfoque puede aplicarse con éxito a paradigmas relacionados, como análisis sintáctico de constituyentes, siempre y cuando se utilicen redes neuronales recurrentes (RNN). En general, este método puede aplicarse a una amplia gama de arquitecturas neuronales. En esta tesis nos apoyamos en arquitecturas basadas en BiLSTM y BERT, que ampliamos con MTL.

Reformular el análisis sintáctico de dependencias como etiquetado secuencial ofrece varias ventajas. En primer lugar, nuestro analizador de etiquetado secuencial se basa en una arquitectura conceptualmente sencilla y prescinde de los algoritmos tradicionales de análisis sintáctico o de las estructuras auxiliares. Esto, a su vez, contribuye a mitigar el recurrente problema de la velocidad en el análisis sintáctico. En segundo lugar, nuestro enfoque tiene otras implicaciones prácticas. Por ejemplo, este método permite ejecutar un analizador sintáctico en un software genérico de etiquetado secuencial, y además provee una manera sencilla de utilizar el árbol de análisis sintáctico completo como entrada a otras tareas de procesamiento de lenguaje natural.

El pilar del análisis sintáctico de dependencias como etiquetado secuencial son las codificaciones que permiten convertir un árbol de dependencias en una secuencia de etiquetas. La principal contribución de nuestro trabajo consiste en presentar tres familias de codificaciones que difieren en la forma de representar un árbol de dependencias. Examinamos empíricamente su viabilidad en términos de precisión y analizamos sus facetas. Además, nuestro trabajo explora el aprovechamiento de datos complementarios externos para aumentar el conocimiento y la competencia de nuestro analizador sintáctico. Para ello, examinamos

la capacidad del analizador sintáctico de etiquetado secuencial para explotar representaciones procedentes de otro paradigma de análisis sintáctico, así como de datos de movimientos oculares.

C.1 Linealizaciones

Las codificaciones desempeñan un papel fundamental en nuestro enfoque, ya que sirven como métodos de linealización que permiten asignar árboles de dependencias a etiquetas discretas. En concreto, en esta tesis nosotros trabajamos con tres familias de codificaciones: (i) *selección de núcleos* que codifica la posición del núcleo directamente en la etiqueta o de forma relativa, (ii) *corchetes* que representa los arcos de dependencias izquierda y derecha en términos de pares equilibrados de elementos de paréntesis y (iii) *transiciones* que divide las transiciones recuperadas de los sistemas basados en transiciones en subsecciones asignadas a cada palabra. Como resultado, cada familia tiene características diferentes que discutiremos a continuación con el apoyo de los resultados empíricos obtenidos con modelos basados en estas codificaciones. Además, cada codificación va acompañada de un método de descodificación que convierte de nuevo una secuencia de etiquetas predicha en un árbol de dependencias. Sin embargo, como algunas de las etiquetas predichas pueden ser erróneas, contribuyendo a crear un árbol malformado, el método de descodificación debe controlar también esta situación. Para ello, definimos heurísticas de postprocesado que garantizan que el analizador sintáctico devuelva un árbol de dependencias acíclico, en el que cada palabra depende únicamente de un único núcleo.

C.1.1 Codificaciones basadas en selección de núcleos

El Capítulo 3 presenta las codificaciones de selección de núcleos en las que cada etiqueta para una palabra dada codifica su núcleo utilizando su posición absoluta en una frase o la distancia relativa entre el núcleo y el dependiente de acuerdo a distintos criterios. En concreto, en esta tesis hemos trabajado con tres codificaciones de este tipo.

En primer lugar, aplicamos *la codificación posicional naif*, donde la etiqueta de una palabra dada contiene directamente el índice de su núcleo y la relación de dependencia correspondiente a ese arco. Como ejemplo, se puede suponer que el núcleo de una palabra w está en el índice 4 con una relación de dependencia *det*, por lo que la etiqueta de la palabra enfocada es de la forma $(4, det)$.

En segundo lugar, utilizamos *la codificación posicional relativa*, que codifica la distancia relativa en términos de la posición del índice. Por ejemplo, una etiqueta $(+1, det)$ en esta codificación denotaría entonces que la palabra precedente es el núcleo de la palabra enfocada. Esta codificación se ha utilizado en trabajos anteriores (Li y col., 2018; Kiperwasser y Ballesteros, 2018).

En tercer lugar, proponemos *la codificación relativa basada en el etiquetado morfológico* que codifica la distancia relativa utilizando el etiquetado morfológico del núcleo de la palabra. Por ejemplo, una etiqueta para una palabra w que contenga $(+1, N, det)$ indica que el núcleo es la primera palabra a la derecha con el etiquetado morfológico N. Esta codificación es similar a la propuesta por Spoustová y Spousta (2010).

Los resultados de los experimentos sugieren que las codificaciones difieren en el espacio de etiquetas que generan. La codificación posicional naif es la que sufre mayor dispersión de etiquetas, lo que contribuye a una precisión considerablemente menor. Por su parte, la codificación relativa basada en en el etiquetado morfológico consigue la mejor precisión. Sin embargo, el rendimiento de esta codificación está condicionado por la calidad y la cantidad de los etiquetados morfológicos disponibles en un corpus. También investigamos si el aprendizaje de esta codificación puede mejorarse aplicando el aprendizaje multitarea, donde cada etiqueta se descompone en sus componentes aprendidos como tareas separadas. Los resultados muestran que el aprendizaje de esta codificación como dos tareas produce ganancias adicionales en la precisión. Además, esta codificación se contradice con los recientes hallazgos que cuestionan la utilidad de los etiquetados morfológicos en el análisis sintáctico de dependencias.

C.1.2 Codificaciones basadas en corchetes

En el Capítulo 4 introducimos otra familia de codificaciones que se basa en corchetes equilibrados. Más en detalle, proponemos la codificación 1-planar en la que un arco de dependencia izquierdo se codifica con un par de corchetes (`<,\`), mientras que un arco de dependencia derecho se codifica con (`/,>`). Nuestros experimentos demuestran que la codificación basada en corchetes 1-planar es viable pero se comporta ligeramente peor que la codificación relativa basada en el etiquetado morfológico en el caso de utilizar los etiquetados morfológicos como características del modelo. Sin embargo, la tendencia se invierte cuando no se utilizan etiquetados morfológicos. Esto puede explicarse por el hecho de que la codificación basada en corchetes no depende de ninguna característica. En resumen, los experimentos sugieren que esta aproximación puede ser una alternativa útil en la ausencia de información morfológica para un determinado idioma. Además, la codificación 1-planar no proporciona una cobertura completa de los árboles no proyectivos. En concreto, los arcos que se cruzan en la misma dirección se descodifican como proyectivos. Esta es una deficiencia con respecto a la codificación basada en selección de núcleos que soporta completamente los árboles no proyectivos. Para paliar esta limitación, proponemos una variante de esta codificación que se basa en una propiedad denominada *2-planaridad*.

En este segundo enfoque los arcos se dividen en dos planos de forma que los arcos que pertenecen al mismo plano no se cruzan. Para ello, seguimos dos estrategias de asignación de planos que evitan el uso del segundo plano siempre que no sea necesario. En la primera estrategia, asignamos un arco a al primer plano si ningún arco que cruce a tiene ya asignado el primer plano. En caso contrario, asignamos el segundo plano siempre que sea posible, es decir, si el arco a cruza arcos asignados a ambos planos, no se asigna a ningún plano. Sin embargo, esta estrategia puede llevar a decisiones subóptimas en la que no se puede asignar un plano a un arco aunque el árbol sea realmente 2-planar. Para garantizar la cobertura completa de los árboles 2-planar, introducimos una segunda estrategia de asignación de planos. En este segundo caso cuando un arco se asigna al primer plano, prohibimos ese plano para sus vecinos (es decir, los arcos que lo cruzan), a continuación prohibimos el segundo plano para los vecinos de sus vecinos, y así sucesivamente. Para utilizar las estrategias en el contexto del análisis sintáctico de dependencias como etiquetado secuencial, introducimos un método que equilibra los corchetes y que codifica los arcos en el segundo plano. Para ello, ampliamos los elementos de los corchetes con elementos adicionales ($B^* = \{<^*, \backslash^*, /^*, >^*\}$) que denotan la pertenencia al segundo plano. Los resultados empíricos muestran que las codificaciones 2-planar son capaces de preservar casi por completo los árboles no proyectivos sin apenas coste adicional en la velocidad de análisis. Además, ambas codificaciones 2-planar proporcionan mejoras en la precisión con respecto a la codificación basada en corchetes 1-planar. Hemos comprobado que las codificaciones 2-planar basadas en diferentes estrategias de asignación de planos tienen un rendimiento similar en términos de precisión, aunque la estrategia basada en la propagación de restricciones tiene una ventaja teórica. Esta ventaja no se tradujo en una mayor precisión, probablemente debido al hecho de que pocos árboles de dependencias no podían ser representados en 2 planos.

C.1.3 Codificaciones basadas en transiciones

En el Capítulo 5 describimos la última familia de linealizaciones con la que trabajamos en esta tesis, que se apoya en los conocidos sistemas basados en transiciones. Esto permite establecer un vínculo teórico entre el análisis sintáctico basado en transiciones y el etiquetado secuencial. Por lo tanto, el objetivo sería encontrar un mapping o una forma de transicionar entre ambos paradigmas. Para ello, utilizamos una *las transiciones de lectura* para dividir la secuencia de transiciones, donde cada una de estas subsecciones corresponderá con la etiqueta para un token dado. Por lo tanto, esto implica que el procesamiento de una frase con n tokens requerirá n tales transiciones de lectura. En este capítulo, se demuestra que que este método es genérico y puede aplicarse a una amplia gama de sistemas left-to-right basados en transiciones que

proporcionan codificaciones obtenidas automáticamente de los algoritmos existentes basados en transiciones. Otra ventaja de este enfoque, aunque queda fuera del ámbito de esta tesis, es que puede extenderse fácilmente a los analizadores de constituyentes basados en transiciones y a los analizadores semánticos.

En este capítulo, también probamos la viabilidad de este enfoque con varias codificaciones basadas en transiciones, en un conjunto diverso de lenguas. En particular, probamos la codificación basada sobre los siguientes algoritmos de transiciones: arc-standard, arc-eager, arc-hybrid y Covington no proyectivo. Demostramos que logran una precisión y una velocidad de análisis sintáctico competitivas. En general, los modelos con codificaciones basadas en arc-standard y arc-hybrid tienden a generar el menor vocabulario de salida, indicando que generan las representaciones más compactas dentro de esta familia de linealizaciones. Sin embargo, también observamos que la codificación basada en el algoritmo no proyectivo de Covington da lugar a una precisión considerablemente menor, a pesar de ser la única codificación que admite la no proyectividad en nuestros experimentos. Esto puede explicarse por el hecho de que este algoritmo no se ejecuta en $\mathcal{O}(n)$ transiciones por frase. Por lo tanto, amplía el tamaño del vocabulario impidiendo un aprendizaje efectivo. Además, mostramos que los modelos con codificaciones basados en transiciones obtienen un rendimiento comparable con los modelos de la familia basada en la selección de núcleos y en corchetes.

C.2 Aprendizaje con datos complementarios

En la segunda parte de la tesis, exploramos si el aprendizaje de nuestro analizador de etiquetado secuencial puede mejorarse con representaciones complementarias extraídas de los datos externos adicionales, manteniendo los objetivos iniciales de la tesis de conseguir una buena velocidad y poder utilizarse de manera genérica por parte de cualquier sistema de etiquetado secuencial. Más en detalle, en esta tesis trabajaremos con representaciones auxiliares procedentes del análisis sintáctico de constituyentes y los datos de seguimiento ocular. Para el aprendizaje conjunto, utilizamos el aprendizaje multitarea en el que los datos complementarios se utilizan como tareas auxiliares. Además, cualquier otro tipo de datos puede aplicarse a nuestro analizador sintáctico siempre que pueda definirse como etiquetado de secuencias, como por ejemplo datos de reconocimiento de entidades nombradas o de chunking.

C.2.1 Aprendizaje conjunto de las representaciones de dependencia y constituyentes

En el Capítulo 6 investigamos métodos para aprovechar la naturaleza complementaria del análisis sintáctico de dependencias y de constituyentes mediante el aprendizaje conjunto de sus representaciones. Para ello, además de nuestro métodos de linearización para análisis el sintáctico de dependencias (en concreto, la codificación relativa basada en el etiquetado morfológico), también usamos la reducción a etiquetado secuencial para análisis sintáctico de constituyentes presentada por Gómez-Rodríguez y Vilares (2018). Probamos los modelos en cuatro configuraciones (i) paradigma único, tarea única (S-S) (ii) paradigma único, multitarea (S-MTL), (iii) paradigma doble, modelos multitarea con pérdidas auxiliares (D-MTL-AUX) y (iv) paradigma doble, multitarea (D-MTL). Más concretamente, en la conficuración S-S entrenamos dos modelos distintos correspondientes a cada uno de los paradigmas de análisis sintáctico que sirven como modelos de referencia. En S-MTL para cada paradigma entrenamos un modelo, en el que cada componente de la etiqueta corresponde a una subtarea distinta. En D-MTL-AUX la predicción de etiquetas parciales para un paradigma de análisis sintáctico se define como las tareas principales, mientras que las etiquetas del paradigma homólogo se consideran tareas auxiliares. En D-MTL todas las etiquetas se aprenden como tareas principales con un factor de ponderación igual. Esto implica que el análisis sintáctico de las dependencias y las constituciones se realiza con un único modelo.

Los resultados empíricos muestran que el uso de datos complementarios como tarea auxiliar del paradigma de análisis sintáctico homólogo (D-MTL-AUX) se traduce en una mayor precisión del paradigma de interés en comparación con un modelo de un solo paradigma (S-S). El enfoque en el que ambos paradigmas se aprenden como tareas principales (D-MTL) resulta en un único modelo que es capaz de encapsular ambas representaciones y de predecir tanto las etiquetas de dependencias como las de constituyentes con una buena precisión. Además, hemos demostrado que los modelos mejorados con datos complementarios no tienen casi ningún coste en términos de velocidad y proporcionan un rendimiento competitivo en comparación con los sistemas existentes.

C.2.2 Uso de datos de seguimiento ocular en el análisis de dependencias

En el Capítulo 7 exploramos el efecto de utilizar datos de seguimiento ocular para entrenar nuestro analizador sintáctico. Para ello, aprovechamos los datos con medidas de los movimientos oculares llamados datos de *seguimiento ocular* que se han aplicado previamente a una serie de tareas en el campo de procesamiento de lenguaje natural

(PLN). La idea principal de utilizar este tipo de datos complementarios es guiar a nuestro analizador a través de los movimientos oculares.

Dado que los datos de seguimiento ocular proporcionan varias medidas de la mirada que pueden utilizarse como características de entrada para un modelo, seguimos trabajos anteriores (Barrett y col., 2016; Hollenstein y Zhang, 2019) y aplicamos 12 características de la mirada agrupadas en cuatro subconjuntos. Sin embargo, no es realista esperar datos de seguimiento ocular para todas las tareas de PLN, por lo tanto, es importante desarrollar técnicas que puedan explotar esos datos durante el entrenamiento, sin requerirlos en el momento de la inferencia. Por ello, en nuestro enfoque las características de la mirada se aprenden como tareas auxiliares durante el entrenamiento, por lo que no son necesarias en el momento de la evaluación. También proponemos un método para entrenar nuestro analizador sintáctico en datos paralelos y disjuntos, ya que el seguimiento ocular todavía no está fácilmente accesible en los corpus de árboles de dependencias. Los datos paralelos contienen anotaciones de dependencias y mirada alineados sobre las mismas oraciones, mientras que los datos disjuntos incluyen datos adicionales no solapados con anotaciones de dependencia.

En general, los resultados, aunque ofrecen mejoras modestas, pueden ser considerados positivos. Sin embargo, los resultados empíricos muestran que hay una divergencia en los resultados entre el conjunto de desarrollo y el de test, lo que sugiere que nuestros modelos pueden carecer de la suficiente capacidad de generalización. Aún así, la principal contribución de esta parte de la tesis es investigar la viabilidad de utilizar características de la mirada para entrenar modelos supervisados, más concretamente en el contexto del análisis sintáctico de dependencias como etiquetado secuencial.

C.3 Conclusiones

En el Capítulo 8 ofrecemos una discusión detallada de nuestro enfoque y planes de trabajo futuro. En resumen, en esta tesis hemos demostrado que el análisis sintáctico de dependencias puede ser reducido a una tarea de etiquetado secuencial, y que este tipo de aproximación es un método viable para entrenar modelos supervisados mediante arquitecturas neuronales. Además, este tipo de aproximación ofrece varias ventajas. Por ejemplo, ofrece un gran equilibrio entre precisión y velocidad. Además, puede aplicarse fácilmente sobre cualquier software de etiquetado de secuencias y utilizarse eficazmente junto con las tareas posteriores.

En concreto, en la tesis hemos propuesto y discutido tres familias de codificaciones basadas en: (i) selección de núcleos, (ii) corchetes y (iii) transiciones. Basándonos en los resultados empíricos, hemos discutido sus ventajas y limitaciones. En conjunto, las familias de codificaciones difieren en la forma en que representan un árbol de dependencias y, en

consecuencia, se asocian con facetas diversas. En lo que respecta a la precisión, la codificación relativa basada en el etiquetado morfológico obtiene el mejor rendimiento, pero únicamente en el caso de utilizar etiquetas morfológicas como parámetros de entrada. Sin embargo, si no se desea utilizar los etiquetadores morfológicos debido a su baja calidad o a la arquitectura del modelo, otras codificaciones son una mejor alternativa. En el caso de un banco de árboles altamente no proyectivo, son preferibles las codificaciones basadas en corchetes 2-planar y la codificación relativa basada en el etiquetado morfológico que cubre completamente los árboles no proyectivos.

Además, hemos demostrado que este tipo de analizadores sintácticos de dependencias pueden ser entrenados con datos auxiliares que proporcionan información complementaria, con el objetivo de obtener así un mejor rendimiento en la tarea objetivo. Esto es especialmente práctico cuando los datos contienen información a nivel de token y, por tanto, pueden incorporarse fácilmente a nuestro marco. En concreto, hemos explorado dos tipos de datos complementarios.

En primer lugar, aprendimos las representaciones del análisis sintáctico de las dependencias y de las constituciones de forma conjunta. En este sentido, los experimentos realizados muestran que los modelos de aprendizaje multitarea sobre estos dos paradigmas, con funciones de pérdida auxiliares para el análisis de constituyentes, pueden superar a los modelos que son entrenados únicamente para realizar análisis de dependencias. Además, también hemos estudiado cómo bajo este paradigma, los análisis sintácticos de dependencias y constituyentes pueden ser aprendidos conjuntamente como tareas principales por un único modelo, sin apenas coste en términos de velocidad y precisión.

En segundo lugar, también hemos utilizado datos de seguimiento ocular para guiar a nuestros analizadores sintácticos de dependencias con representaciones de los movimientos oculares. Para ello, los rasgos de la mirada se aprendieron de nuevo como tareas auxiliares, de manera que en tiempo de ejecución dichos datos no necesitan ser proporcionados como entrada. Se optó por esta aproximación dado que asumir que dichos datos oculares están disponibles ampliamente para cualquier texto de entrada es poco realista. En general, los resultados empíricos son modestos pero positivos, lo que sugiere que el aprendizaje de características de la mirada como tareas auxiliares puede ser beneficioso para los analizadores sintácticos de etiquetado de secuencias. No obstante, los resultados motivan a seguir investigando sobre el uso de datos de procesamiento del lenguaje humano en el análisis sintáctico de las dependencias.

C.3.1 Trabajo futuro

Los estudios futuros deberían tener como objetivo la exploración de otras arquitecturas de redes neuronales profundas recientes que puedan

mejorar aún más la precisión del analizador de etiquetado de secuencias y cerrar la brecha con los analizadores de dependencias más avanzados, manteniendo el equilibrio entre la velocidad de análisis y la precisión. En cuanto a los métodos de codificación, las investigaciones futuras no deberían limitarse a los propuestos en este trabajo. En el caso de las codificaciones de selección de núcleos, se puede aplicar cualquier método alternativo para codificar los núcleos, por ejemplo, utilizando características morfológicas. Con respecto a la codificación basada en corchetes de 2-planar, hemos propuesto dos estrategias de asignación de planos que minimizan el uso del segundo plano. Aunque se pueden explorar otras estrategias basadas en diferentes criterios. Además, los algoritmos basados en transiciones existentes ofrecen una serie de codificaciones prometedoras que pueden obtenerse automáticamente y aplicarse en el análisis sintáctico del etiquetado de secuencias. Este método puede también ampliarse y aplicarse a los sistemas de transiciones para análisis sintáctico de constituyentes y análisis semántico.

Además, los resultados basados en los datos complementarios son alentadores, por lo que se necesitan estudios futuros para explorar otros tipos de datos complementarios que puedan ser beneficiosos en el análisis sintáctico de dependencias. En el contexto de los datos de seguimiento ocular, se empiezan a utilizar otros tipos de mediciones en PLN, como el electroencefalograma (Zhang y col., 2018) o la resonancia magnética funcional (Bingel, Barrett y Søgaard, 2016; Toneva y Wehbe, 2019), y surgen nuevas directrices para maximar los beneficios del uso de datos de seguimiento ocular (Hollenstein, Barrett y Beinborn, 2020). Aún así, hay margen de mejora, por ejemplo, en el aumento de la capacidad de generalización de nuestro modelo para mitigar la divergencia en los resultados entre el conjunto de desarrollo y el de test. Para ello, se puede considerar la aplicación de otra arquitectura, por ejemplo, siguiendo Barrett y col. (2018) los datos de eye-tracking podrían servir para regularizar la atención. Sin embargo, eso tendría un cierto coste en cuanto a velocidad.