

DEEP LEARNING-BASED DEPENDENCY PARSING FOR TURKISH

by

Şaziye Betül Özateş

B.S., Computer Engineering, Boğaziçi University, 2012

M.S., Computer Engineering, Boğaziçi University, 2014

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Computer Engineering
Boğaziçi University

2022

*to my late grandfather Ziya Bilgin,
who always encouraged me to pursue knowledge.*

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to my advisors Assoc. Prof. Arzucan Özgür and Prof. Tunga Güngör for their endless support and for their kindness, patience, and motivation throughout my PhD studies. I couldn't ask for better advisors and I will be forever grateful for their guidance in this journey.

I would also like to thank Prof. Balkız Öztürk Başaran for her guidance in linguistics aspects of my thesis and for improving my knowledge in this area. I would like to thank the rest of my thesis committee: Prof. Banu Diri, Prof. Deniz Yüret, and Assist. Prof. Berk Gökberk for their insightful and valuable comments and encouragement.

My sincere thanks also goes to the professors of the Computer Engineering Department of Boğaziçi University. I am beyond grateful to Dr. Suzan Üsküdarlı for her unlimited support since my undergraduate years and for always giving her positive energy. I am thankful to Prof. Ethem Alpaydın, Prof. Alper Şen, and Prof. Ali Taylan Cemgil for their wonderful lectures I had the chance to attend throughout my studies.

I thank to Dr. Özlem Çetinoğlu whom I had the chance to work with at the University of Stuttgart. I am grateful to her for enriching my knowledge in the notion of code-switching, and also for being a cheerful friend during our working days together.

I thank my labmates in TABILAB for insightful discussions and for answering all of my questions. I owe a huge thank to Hakime Öztürk for being my support system. Also I thank my friends in the Linguistics Department whom I worked with together.

I could not complete this thesis without the moral support of my friends and family: dearest Şeyma, Beyza, and many other beloved ones not mentioned by name. Many thanks to my in-laws, Halide and Veysel Özateş who supported me in so many ways throughout my thesis study.

Last but not the least, I would like to thank my dearest parents Güzide Bilgin and Abdullah Bilgin who always show their endless love and unconditional support in every step of my life and encourage me to pursue an academic career. I am beyond thankful to my sisters Esma, Rümeysa, and Zehra, and to my brother Ahmed for being my best friends and for motivating and guiding me in this challenging journey. I want to express my gratitude to my beloved husband Mustafa Erkam for always believing in me and supporting me in every way and to my dearest son Harun for being the greatest joy of my life.

I gratefully acknowledge the financial support provided by the Scientific and Technological Research Council of Turkey (TÜBİTAK) under grant number 117E971 and as BİDEB 2211 graduate scholarship.

ABSTRACT

DEEP LEARNING-BASED DEPENDENCY PARSING FOR TURKISH

Dependency parsing is an important step for many natural language processing (NLP) systems such as question answering and machine translation. Turkish, being a morphologically rich language and having a complex grammar, is challenging for automatic processing. Limited NLP tools and resources for Turkish make the task even more challenging. Data-driven deep learning models show promising performance in dependency parsing. Yet, the amount of data to train a data-driven dependency parser directly affects performance, and deep learning-based systems require extensive data to achieve good performance. In this thesis, we focused on Turkish dependency parsing and proposed two solutions to the challenges this task poses. First, we increased the size and quality of labeled data for Turkish dependency parsing. In this respect, we created the BOUN Treebank by annotating 9,761 sentences. In addition, we re-annotated the IMST and PUD treebanks using the same annotation scheme. As a result, we presented the largest collection of Turkish treebanks with consistent annotation. Second, we developed novel state-of-the-art dependency parsing models for Turkish as well as other low-resource languages. As our first parsing approach, we introduced a hybrid dependency parser where Turkish grammar rules and morphological features of words are integrated into the deep learning model. Despite the limited training data, the hybrid parser achieved higher success than the current methods for Turkish dependency parsing. As our second parsing approach, we proposed a deep dependency parser with semi-supervised enhancement. By conducting experiments on a number of low-resource languages besides Turkish, we achieved state-of-the-art results on all datasets. We have shown that deep learning-based models can be improved not only by additional training data, but also by integrating intelligently extracted information.

ÖZET

DERİN ÖĞRENME TABANLI TÜRKÇE BAĞLILIK AYRIŞTIRMASI

Bağlılık ayrıştırma, otomatik soru cevaplama ve makine çevirisi gibi birçok doğal dil işleme (DDİ) sistemi için önemli bir adımdır. Zengin morfolojisi ve karmaşık gramer yapısıyla Türkçe dili otomatik işlenmesi oldukça zor bir dildir. Türkçe DDİ araçlarının ve kaynaklarının kısıtlı olması bu işi daha da zorlaştırmaktadır. Veri güdümlü derin öğrenme modelleri, bağlılık ayrıştırma alanında etkili performans göstermektedir. Veri güdümlü bir bağlılık ayrıştırıcıyı eğitmek için gereken verinin miktarı ayrıştırıcının performansını doğrudan etkilemektedir. Ayrıca, derin öğrenme tabanlı sistemlerin yüksek başarı göstermesi için büyük miktarlarda veriye ihtiyaç duyduğu gözlemlenmiştir. Bu tezde, Türkçe bağlılık ayrıştırmadaki zorlukların üstesinden gelmek için iki tip çözüm önerdik. İlk olarak, Türkçe metinleri ayrıştırmak için gereken veri miktarını ve kalitesini artırdık. Bu bağlamda, 9.761 yeni cümleyi manuel olarak etiketleyerek BOUN ağaç yapılı derlemi oluşturduk. Aynı etiketleme şemasıyla IMST ve PUD ağaç yapılı derlemelerini de yeniden etiketledik. Bu sayede Türkçe için dil bilgisi kurallarına göre tutarlı en büyük ağaç yapılı derlem koleksiyonunu kullanıma sunduk. İkinci olarak, Türkçe ve diğer az kaynaklı diller için özgün ve son teknoloji bağlılık ayrıştırıcılar geliştirdik. Önce, Türkçe dil bilgisi kurallarının ve kelimelerin morfolojik özelliklerinin derin öğrenme modeline entegre edildiği bir hibrit bağlılık ayrıştırma mimarisi önerdik. Sınırlı eğitim verisine rağmen, hibrit ayrıştırıcıyla Türkçe bağlılık ayrıştırmada mevcut yöntemlerden daha yüksek başarı elde ettik. Ayrıca, yarı denetimli geliştirmeye dayalı bir derin öğrenme tabanlı bağlılık ayrıştırıcı önerdik. Türkçe'nin yanı sıra kaynak yetersizliği olan başka dillerde de deneyler yaparak son teknoloji sonuçlar elde ettik. Derin öğrenme tabanlı modellerin yalnızca fazla miktarda eğitim verisiyle değil, aynı zamanda akıllıca çıkarılan bilgilerin entegrasyonu ile da geliştirilebileceğini gösterdik.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	iv
ABSTRACT	vi
ÖZET	vii
LIST OF FIGURES	xiii
LIST OF TABLES	xv
LIST OF SYMBOLS	xviii
LIST OF ACRONYMS/ABBREVIATIONS	xix
1. INTRODUCTION	1
1.1. Problem Statement	1
1.2. Motivation and Objective	4
1.3. Publication Notes	5
1.4. Thesis Overview	6
2. LINGUISTIC BACKGROUND	9
2.1. Turkish	9
2.2. Code-Switching	11
3. THEORETICAL BACKGROUND	13
3.1. Dependency Parsing Approaches	13
3.1.1. Grammar-driven Methods	13
3.1.1.1. Context-free-based Methods	13
3.1.1.2. Constraint-based Methods	13
3.1.2. Data-driven Methods	13
3.1.2.1. Transition-based Methods	14
3.1.2.2. Graph-based Methods	14
3.2. Related Deep Learning Methodologies	14
3.2.1. Long Short-Term Memory Networks	15
3.2.1.1. Stack-LSTM	15
3.2.1.2. Bidirectional LSTMs	16
3.2.2. Transformer-based Language Models	16

3.2.2.1.	BERT	16
3.2.2.2.	XLM-R	16
3.3.	Evaluation Metrics	17
4.	THE BOUN TREEBANK: A NEW AND HIGH QUALITY CORPUS FOR TURKISH DEPENDENCY PARSING	18
4.1.	Introduction	18
4.2.	Related Work	19
4.3.	The BOUN Treebank	24
4.3.1.	Levels of Annotation	26
4.3.1.1.	Morphology	26
4.3.1.2.	Syntax	29
4.3.2.	Different Conventions Adopted in the Annotation Process	31
4.3.2.1.	Annotation of Embedded Clauses	32
4.3.2.2.	Copular Clitic	35
4.3.2.3.	Compound	37
4.3.2.4.	Classifier	40
4.3.2.5.	Core Arguments	41
4.3.2.6.	Summary of the linguistic considerations	44
4.4.	Experiments	44
4.5.	Results	46
4.5.1.	Parsing Results on the BOUN Treebank	46
4.5.2.	Parsing Results on Combinations of Treebanks	48
4.5.3.	Changes in Dependency Label Distribution	50
4.6.	Conclusion	53
5.	A MORPHOLOGY-BASED REPRESENTATION MODEL FOR DEEP DE- PENDENCY PARSING	54
5.1.	Introduction	54
5.2.	The Parsing Model	54
5.2.1.	Character Embeddings of Words	55
5.2.2.	Morphology-based Character Embeddings	55
5.2.2.1.	Lemma-Suffix Model	56

5.2.2.2.	Morphological Features Model	57
5.3.	Experiments	58
5.3.1.	Extracting Lemmas and Suffixes	59
5.3.2.	Embedding Model Selection for Different Languages	59
5.3.3.	Training Specifications	61
5.4.	Results	62
5.5.	Conclusion	65
6.	A HYBRID DEEP DEPENDENCY PARSER ENHANCED WITH RULES AND MORPHOLOGY	66
6.1.	Introduction	66
6.2.	Related Work	67
6.3.	Methods	70
6.3.1.	Stanford’s Neural Dependency Parser	71
6.3.2.	A Rule-based Unlabeled Parsing Approach	71
6.3.3.	Integrating the Rule-based Approach with Stanford’s Graph- based Parsing Method	84
6.3.4.	A Morphology-based Enhancement for Dependency Parsing	85
6.3.4.1.	The Inflectional Suffixes Model	86
6.3.4.2.	The Last Suffix Model	86
6.3.4.3.	The Suffix Vector Model	86
6.4.	Experiments	88
6.5.	Results	90
6.5.1.	Ablation Study	90
6.5.2.	Comparison of the Models	93
6.5.3.	Parsing Performance on Raw Text	96
6.5.4.	Error Analysis on the Rules	97
6.5.5.	The Effect of Training Data Size	100
6.6.	Generalization to Other Languages	104
6.6.1.	Adapting the Rule-based Parsing Approach	104
6.6.2.	Adapting the Morphology-based Enhancement Approach	105
6.7.	Conclusion	105

7. A SEMI-SUPERVISED DEPENDENCY PARSER FOR LOW-RESOURCE LANGUAGES: A CASE STUDY FOR CODE-SWITCHED TEXTS	107
7.1. Introduction	107
7.2. Related Work	109
7.3. Methods	110
7.3.1. Base Parsing Model	110
7.3.2. Semi-supervised Enhancement through Auxiliary Sequence Labeling Tasks	111
7.3.3. The Gating Procedure	114
7.3.4. Transformer-based Adaptation of the Model	115
7.4. Experiments	115
7.4.1. Data	115
7.4.2. Training Setup	116
7.4.2.1. Unlabeled Data	116
7.4.2.2. Sequence Labeler Training	117
7.4.3. Baselines	118
7.4.4. Semi-supervised Enhancement Models	119
7.5. Results	120
7.5.1. Parsing of Code-Switching Language Pairs	120
7.5.2. Comparison of Methods in terms of Computational Resources	124
7.5.3. Effect of Gold Labeled Data on the Parsing Performance	125
7.5.4. Parsing of Turkish	126
7.6. Conclusion	128
8. TOOLS	129
8.1. BoAT Annotation Tool	129
8.1.1. Features	130
8.1.2. Implementation	133
8.2. BOUN-PARS	133
8.2.1. Requirements	134
8.2.2. Usage of BOUN-Pars	134
8.2.3. Reproducing the Original Results	135

9. CONCLUSION	136
9.1. Summary of Contributions	136
9.2. Future Work	139
REFERENCES	140

LIST OF FIGURES

Figure 1.1.	Dependency tree of an example sentence.	1
Figure 2.1.	Dependency tree of a code-switched sentence from the Turkish-German SAGT Treebank.	12
Figure 4.1.	The average token count and the average dependency arc length of the BOUN Treebank.	47
Figure 5.1.	Vector representation with the character-based embedding model.	55
Figure 5.2.	Character-based word embedding using Lemma-Suffix Embedding Model.	57
Figure 5.3.	Character-based word embedding using Morphological Features Embedding Model.	58
Figure 6.1.	Our rule-based dependency parser.	72
Figure 6.2.	Operation of the linguistic rules on a sentence.	83
Figure 6.3.	The word embedding representation of the hybrid model with the rule-based enhancement.	85
Figure 6.4.	Dense representation of the word <i>insanların</i> (<i>people's</i>) using the Inflectional Suffixes Model.	87
Figure 6.5.	Dense representation of the word <i>insanların</i> (<i>people's</i>) using the Last Suffix Model.	87

Figure 6.6.	Word representation model of the system with the Suffix Vector Model.	87
Figure 6.7.	A small subset of the lemma-suffix matrix created from the Newscor part of the Boun Web Corpus.	88
Figure 6.8.	The effect of each rule to the parsing performance on the IMST Treebank.	92
Figure 6.9.	The effect of each rule to the parsing performance on the IMST Treebank, when AdvVerb and NounVerb rules are removed from the rule-based parsing system.	92
Figure 6.10.	Confusion matrices for detecting complex predicates, noun compounds and possessive constructions, adverb-adjective relations, and adjective-noun relations.	99
Figure 6.11.	Effect of increasing the training data size on parsing performance of our hybrid model and the baseline model.	103
Figure 7.1.	Parser architecture with semi-supervised auxiliary task enhancement.	111
Figure 7.2.	Dependency tree of an example sentence from Hindi-English HIENCS Treebank.	114
Figure 7.3.	Comparison of Base _{LSTM} , Self-training, and our best model on Hi-En and Fy-Nl.	125
Figure 8.1.	A screenshot from the BoAT tool.	131

LIST OF TABLES

Table 2.1.	Possible morphological analyses of the word <i>alm</i>	10
Table 4.1.	Comparison of the BOUN Treebank to previous monolingual Turkish treebanks.	23
Table 4.2.	Composition of the written component of TNC using words as the measurement unit.	24
Table 4.3.	Sentence and word statistics for the different sections of the BOUN Treebank.	25
Table 4.4.	The performance of two morphological taggers on BOUN Treebank.	27
Table 4.5.	Mappings of morphological features from the morphological analysis tool to the features in UD framework.	28
Table 4.6.	Example sentence from the BOUN Treebank encoded in CoNNL-U format.	29
Table 4.7.	The dependency relation set of the BOUN Treebank.	30
Table 4.8.	Division of the BOUN Treebank and its different sections among training, development, and test sets	45
Table 4.9.	Attachment scores of the parser on the BOUN Treebank.	46
Table 4.10.	The performance of the parser on five different test sets.	48

Table 4.11.	Dependency label distribution of the BOUN Treebank together with previous and re-annotated versions of IMST and PUD treebanks. .	52
Table 5.1.	Number of occurrences of some example suffixes and corresponding dependency labels of verbs with these suffixes in the development data of the IMST Treebank.	56
Table 5.2.	Lemma and suffix separation example with and without using morphological analyzer and disambiguator.	59
Table 5.3.	List of morphological features for languages with the Morphological Features Embedding Model.	60
Table 5.4.	Comparison of Lemma-Suffix and Morphological Features models with the baseline.	63
Table 5.5.	Word-morpheme structure of Hungarian word <i>ember</i> and Turkish word <i>adam</i>	64
Table 5.6.	Effect of using pretrained word embeddings on parsing of Turkish IMST test set.	65
Table 6.1.	Summary of rules in the rule-based system with examples for each encoding.	82
Table 6.2.	The ablation study steps for the rule-based parser.	91
Table 6.3.	Attachment scores of the baseline parser, proposed hybrid models, and Udify on the IMST Treebank.	93

Table 6.4.	Comparison of our best hybrid model on the IMST test set with the top performing parsing systems in CoNLL 2018 shared task.	95
Table 6.5.	Analysis of the proposed methods on the IMST test set.	98
Table 7.1.	Some statistics and related resources for the CS treebanks.	116
Table 7.2.	Attachment scores of baselines, our models, and the previous works on all CS UD treebanks.	121
Table 7.3.	Performance of XLM-R-based parser and our XLM-R adaptation of auxiliary task enhancement models.	123
Table 7.4.	Comparison of baselines and the proposed approach according to training time, memory usage during training, and LAS.	124
Table 7.5.	Attachment scores of LSTM-based baselines and our models on the test set of the Turkish IMST Treebank.	127
Table 7.6.	Attachment scores of XLM-R-based baselines and our models on the test set of the Turkish IMST Treebank.	127

LIST OF SYMBOLS

b	Bias vectors
c	Cell activation vectors
E_{AT}	Sequence labeler encoder
$e^{labeler}$	Output of the sequence labeler encoder
E_p	Parser encoder
e^{parser}	Output of the parser encoder
E_S	Set of directed edges in G_S
f	Forget gate
G_S	Directed connected graph of sentence S
i	Input gate
h_t	Hidden state at time step t
o	Output gate
S_{arc}	Arc score
S_{label}	Label score
V_S	Set of nodes in G_S
W	Weight matrix
W^{gate}	A learned parameter of the gating procedure
w^{gate}	A learned parameter of the gating procedure
x_t	Input instance at time step t
y_t	Output instance at time step t
σ	Logistic sigmoid function

LIST OF ACRONYMS/ABBREVIATIONS

AdjNoun	Adjective-Noun
AdvAdj	Adverb-Adjective
AdvVerb	Adverb-Verb
BERT	Bidirectional Encoder Representations from Transformers
BiLSTM	Bidirectional Long Short-Term Memory
BLEX	Bi-lexical Dependency Score
BOUN	Turkish BOUN Treebank
BoAT	BoAT Annotation Tool
CMI	Code-mixing Index
ConsecAdj	Consecutive Adjective
ConsecAdv	Consecutive Adverb
CS	Code-Switching
CmpxPred	Complex Predicates and Verbal Idioms
DTR	Distance to the Root
FN	False Negatives
FP	False Positives
Fy-Nl	Frisian-Dutch Fame Treebank
ID	Identification
IMST	Turkish IMST Treebank
Hi-En	Hindi-English HIENCS Treebank
K	Thousand
Kpv-Ru	Komi-Zyrian IKDP Treebank
LAS	Labeled Attachment Score
LIH	Language ID of Head
LID	Language ID
LSTM	Long Short-Term Memory
M	Million
MLAS	Morphology-aware Labeled Attachment Score

MLP	Multi-layer Perceptron
MST	Maximum Spanning Tree
NLP	Natural Language Processing
NOC	Number of Children
NounComp	Noun Compounds
NounVerb	Noun-Verb
OSV	Object-Subject-Verb
OVS	Object-Verb-Subject
PC	Punctuation Count
POS	Part-of-speech
PossConst	Possessive Construction
PUD	Turkish Parallel UD Treebank
RNN	Recurrent Neural Networks
RPE	Relative POS-based Encoding
SMH	Simplified Morphology of Head
SOTA	State-of-the-art
SOV	Subject-Object-Verb
SVO	Subject-Verb-Object
TDK	Turkish Language Association
TN	True Negatives
TNC	Turkish National Corpus
TP	True Positives
Tr-De	Turkish-German SAGT Treebank
UAS	Unlabeled Attachment Score
UD	Universal Dependencies
VOS	Verb-Object-Subject
VSO	Verb-Subject-Object
XLM-R	Cross Lingual Model - Roberta

1. INTRODUCTION

1.1. Problem Statement

Dependency grammar is a way of representing syntactic dependencies between words in sentences. Words are linked to each other with directed links. These links between words are called *dependencies*. Usually, the main verb in a sentence is the root node in the dependency tree of the sentence. Each dependency is directed from the node of the dependent word towards the node of the head word. Figure 1.1 depicts the dependency tree of an example sentence. Each word in a dependency tree is dependent to another word with its corresponding dependency relation type.

Dependency parsing is useful for many natural language processing tasks such as relation extraction, machine translation, and question answering. Using dependency tree representations of sentences, structural information can easily be extracted. Since a dependency tree of a sentence consists of a set of dependent-head pairs, the order of words in the sentence is not important and does not alter construction of the tree. With this advantage, dependency grammars are more suitable than constituency grammars for languages with free word order such as Turkish, Czech, and Finnish [1].

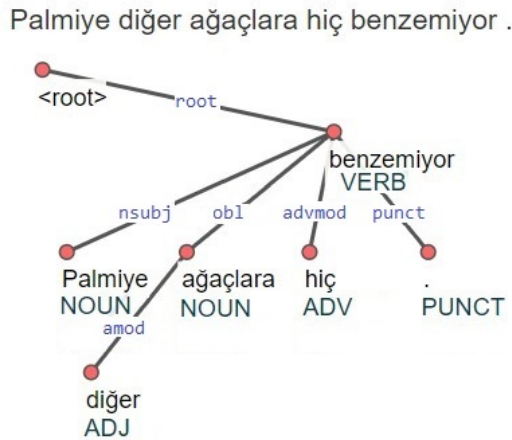
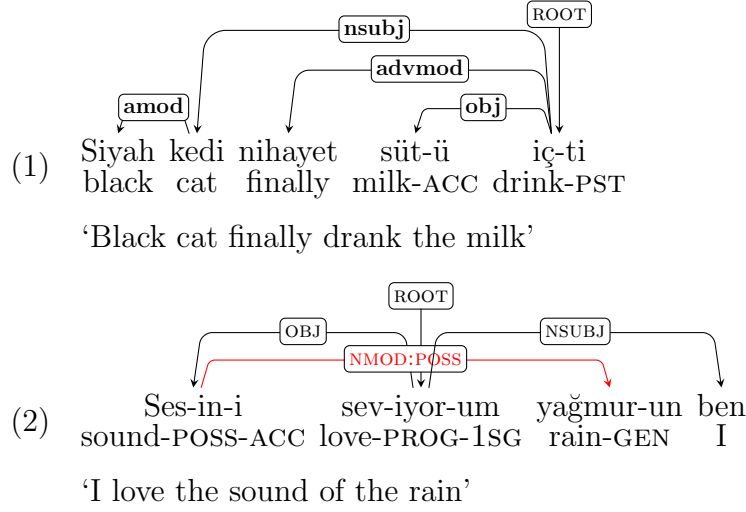


Figure 1.1. Dependency tree of the sentence “Palmiye diğer ağaçlara hiç benzemiyor.” (*Sentence translation: Palm tree is not like any other tree.*)

There are a number of challenges in Turkish dependency parsing. The most prominent of these challenges are stated below:

- *Turkish is a morphologically rich language.* Theoretically there is no limit for adding derivational suffixes to a Turkish word. So, one word in Turkish language may correspond to a whole sentence in other languages like English. The complex structure of such words makes the task of dependency parsing more complex for Turkish.
- *Natural language processing (NLP) of Turkish is not counted as rich in terms of resources available.* For a very long time, the largest dataset used for training and evaluation of dependency parsers was the ITU-METU-Sabancı (IMST) Turkish Treebank [2] which consists of 5,635 grammatically parsed sentences and is currently [3] ranked as the 88th out of 217 treebanks in terms of the number of annotated sentences in the Universal Dependencies (UD) project [4]. The amount of data to train a data-driven dependency parser directly affects the performance of the parser. It is observed that deep learning based systems need large amounts of data to be able to show good performance [5]. So, there is a need to increase the amount of syntactically annotated data for a deep learning-based dependency parser to reach the state-of-the-art performance.
- *Turkish treebanks include substantial amount of non-projective dependencies.* Examples 1 and 2 show a projective dependency tree and a non-projective dependency tree, respectively. In the non-projective example, the dependency edges cannot be drawn in the plane above the sentence without any two edges crossing each other. However, in the projective example, the dependency edges can be drawn in this manner with no edges crossing. Most of the previous systems for Turkish dependency parsing make the projectivity assumption, that is there are no crossing relations in a dependency tree. Yet, it is stated in [6] that more than 20% of the sentences in Turkish have non-projective dependency trees. We need to handle both the projective and non-projective dependency trees in order to increase the success rates.



- *Accuracy of morphological analyzers affects the performance of dependency parsers.*

A study on the impact of using automatic morphological analysis and disambiguation tools in Turkish dependency parsing states that usage of these tools instead of manually performing morphological analysis and disambiguation causes a significant decline in the performance of the parser [7]. Another study observed that using a morphological disambiguator trained with more accurate data increases the success rates of dependency parsing for Turkish [8]. Accuracies of morphological analysis and disambiguation tools also pose a challenge for dependency parsing of Turkish.

This thesis aims to improve the dependency parsing of Turkish language. We approach the problem in two ways: first by developing a successful parsing system to increase the performance in dependency parsing of Turkish texts, and second by increasing the quality and quantity of training data with a new comprehensive treebank manually annotated in Turkish. By making the tools and resources we develop publicly available, we hopefully enable many language processing applications that need dependency parsing to be able to achieve better performance for Turkish. We anticipate that this work will be a base for many new natural language processing studies in Turkish.

1.2. Motivation and Objective

Many applications we use in our daily lives from search engines to recommendation systems utilize natural language processing techniques in background to analyze natural texts. In order for these applications to work successfully in real life, the underlying NLP techniques must achieve high performance.

Automatic syntactic parsing of natural texts is the first main step in the structural analysis of text data. Syntactic parsing of a sentence is finding the correct parse tree that shows the syntactic relations between words, given a grammar. Recently, dependency parsing gains much more attention in NLP studies due to its being computationally more efficient than constituency parsing.

Turkish, being a morphologically rich and free word order language, is not a widely studied language unlike frequently studied ones such as English. The amount of NLP resources for Turkish is restricted and performance of the current state-of-the-art dependency parsers is far from the desired level for Turkish.

Our main objective is creating a state-of-the-art dependency parsing architecture for Turkish to be used in a wide range of NLP applications. Although we give priority to Turkish language and desire to handle the NLP problems of Turkish in the first place, our proposed parsing approaches can easily be adapted to be able to work on other languages.

We also aim to enrich Turkish NLP resources by building a dependency treebank that is superior than the existing Turkish treebanks in terms of quality and quantity. To achieve this aim successfully, collaborating with linguists is essential. As there are only a handful of annotation tools all having their own drawbacks, developing an annotation tool that will exactly meet the demands of annotators for a faster and easier annotation is also one of the requirements of this aim.

Finalizing the end products of scientific researches and releasing them as publicly available is a step that is often neglected. We intend to create a downloadable and online tool for the developed dependency parser and make our treebank and software publicly available for academic use. By this way, natural language processing applications that need dependency parsing will be able to achieve better performance for Turkish.

1.3. Publication Notes

Some parts of this thesis have appeared in the following publications:

- (i) “A morphology-based representation model for LSTM-based dependency parsing of agglutinative languages.” Özateş, Ş. B., Özgür, A., Güngör, T., and Öztürk, B. In Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, (2018), pp. 238-247. (Chapter 4)
- (ii) “Resources for Turkish dependency parsing: Introducing the BOUN treebank and the BoAT annotation tool.” Türk, U., Atmaca, F., Özateş, Ş. B., Berk, G., Bedir, S. T., Köksal, A., Öztürk Başaran, B., Güngör, T., and Özgür, A. Language Resources and Evaluation 56, no. 1 (2022): 259-307. (Chapter 5)
- (iii) “A Hybrid Deep Dependency Parsing Approach Enhanced with Rules and Morphology: A Case Study for Turkish.” Özateş, Ş. B., Özgür, A., Güngör, T., and Öztürk, B. ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP), *under review*. (Chapter 6)
- (iv) “Improving Code-Switching Dependency Parsing with Semi-Supervised Auxiliary Tasks.” Özateş, Ş. B., Özgür, A., Güngör, T., and Çetinoğlu, Ö. Findings of the Association for Computational Linguistics: NAACL 2022, (2022). (Chapter 7)

Some of our other works that are not part of this thesis are:

- (i) “Improving the annotations in the Turkish universal dependency treebank.” Türk, U., Atmaca, F., Özateş, Ş. B., Başaran, B. Ö., Güngör, T., and Özgür, A. In Pro-

ceedings of the Third Workshop on Universal Dependencies (UDW, SyntaxFest 2019), (2019), pp. 108-115.

- (ii) “Turkish treebanking: Unifying and constructing efforts.” Türk, U., Atmaca, F., Özateş, Ş. B., Köksal, A., Başaran, B. Ö., Güngör, T., and Özgür, A. In Proceedings of the 13th Linguistic Annotation Workshop, (2019), pp. 166-177.
- (iii) “A Language-aware Approach to Code-switched Morphological Tagging.” Özateş, Ş. B. and Çetinoğlu, Ö. In Proceedings of the Fifth Workshop on Computational Approaches to Linguistic Code-Switching (CALCS), Association for Computational Linguistics, (2021), pp. 72–83.
- (iv) “Enhancements to the BOUN Treebank Reflecting the Agglutinative Nature of Turkish.” Marşan, B., Akkurt, S. F., Şen, M., Gürbüz, M., Özateş, Ş. B., Üsküdarlı, S., Özgür, A., Güngör, T., Öztürk, B. International Conference on Agglutinative Language Technologies as a challenge of Natural Language Processing (ALTNLP), (2022), accepted.

1.4. Thesis Overview

In this thesis, we focused on improving dependency parsing of Turkish. To achieve this purpose, we advanced our efforts in two directions.

On the one hand, we annotated a new comprehensive corpus for Turkish dependency parsing as a solution to the problem of insufficient data. This new treebank further benefits the parsing performance, promising to be a new base for many new natural language processing studies in Turkish.

On the other hand, we developed novel parsing methods by utilizing deep learning-based techniques and the linguistic structure of the language. This hybrid parsing approach obtained state-of-the-art parsing performance on Turkish, proving the positive effect of morphology on Turkish dependency parsing. Furthermore, we did not only focus on Turkish dependency parsing but we also expanded the focus of the thesis by proposing a novel parsing method with semi-supervised enhancement for low-resource

languages. We experimented with this parser on both Turkish and low-resource code-switching language pairs and reached state-of-the-art performance on all datasets.

In the course of this thesis, we achieved the following contributions:

- (i) A novel hybrid dependency parsing model for Turkish is developed. For this purpose, deep learning techniques and linguistics-based approaches are employed to predict the possible parse trees. State-of-the-art dependency parsing scores are reached on Turkish (Chapter 6) [9].
- (ii) Within the hybrid parsing model, a novel rule-based enhancement method that can be integrated to any neural dependency parser and a morphology-based enhancement method with three different ways of including morphological information to the parser are presented. In addition, a simple yet useful integration method that allows to combine the proposed enhancement methods with any neural dependency parser is introduced (Chapter 6.3) [9].
- (iii) An advanced semi-supervised dependency parser is created that shows state-of-the-art performance especially for low-resource languages. For this purpose, an LSTM-based state-of-the-art dependency parser is enhanced with novel auxiliary tasks trained in a semi-supervised scheme. This parser is evaluated on code-switching language pairs and on Turkish. The proposed approach is also adapted to a transformer-based parser. A comparative study elaborating the effectiveness of both models are presented through several experiments (Chapter 7) [10].
- (iv) A large syntactically annotated corpus consisting of the dependency parse trees of Turkish sentences is created and made publicly available. This corpus has morphologically analyzed sentences of different genres of texts. A large Turkish corpus consisting of dependency parse trees of sentences is beneficial in developing solutions to various NLP problems. In addition, this treebank can be useful for researchers of linguistics in identifying different aspects of the language grammar (Chapter 5) [11].
- (v) The treebank annotation guidelines that are followed to create our dependency treebank are provided and made publicly available. These guidelines explain our

linguistic decisions and annotation scheme that are based on the UD framework. They also demonstrate examples for the challenging issues that are present in the newly created treebank as well as other existing treebanks that we re-annotated. This guideline will hopefully be useful for the creation and enhancement of the treebanks of other languages as well as Turkish.

- (vi) An annotation tool for dependency parsing named BoAT [12], an online tool for the hybrid parser [13], as well as the source codes and trained models for both the hybrid parser [14] and the semi-supervised deep parser [15] are made available for public use.

2. LINGUISTIC BACKGROUND

In this thesis, we apply dependency parsing both to Turkish and to a number of code-switching language pairs as low-resource languages. Sections 2.1 and 2.2 present brief background information on these languages.

2.1. Turkish

This section is based on the Turkish Section of our collaborative work published as [11].

Turkish is a Turkic language spoken mainly in Asia Minor and Thracia with approximately 75 million native speakers. As an agglutinative language, Turkish makes excessive use of morphological concatenation. According to Bickel and Nichols [16], a Turkish verb may have up to 8-9 inflectional categories per word, such as number, tense, or person marking. This number is about twice of the average of the maximum number of inflectional categories in the other 145 languages covered in [16]. The number of morphological categories increases further when considering derivational processes. Kapan [17] states that Turkish words may host up to 6 different derivational affixes at the same time. The complexity of morphological analysis, however, is not limited to the sheer number of inflectional and derivational affixes. In addition to such affixes, allomorphies, vowel harmony processes, elisions, and insertions create a difficult task for researchers in Turkish NLP. Table 2.1 lists the possible morphological analyses of the surface word *alın*. Despite the shortness of the word, the morphological analysis is complicated; even such a short word may be parsed to have different possible roots.

Table 2.1. Possible morphological analyses of the word *alın* from [18]. The symbol ‘&’ in the Features column indicates derivational morphemes and ‘+’ indicates inflectional morphemes.

Root	Category of the root	Features	Gloss	Translation
<i>alın</i>	[Noun]	+ [A3sg] + [Pnon] + [Nom]	forehead	‘forehead’
<i>al</i>	[Noun]	+ [A3sg] + Hn[P2sg] + [Nom]	red-POSS	‘your red color’
<i>al</i>	[Adj]	& [Noun] + [A3sg] + Hn[P2sg] + [Nom]	red-POSS	‘your red color’
<i>al</i>	[Noun]	+ [A3sg] + [Pnon] + NHn[Gen]	red-GEN	‘belonging to the color red’
<i>al</i>	[Adj]	& [Noun] + [A3sg] + [Pnon] + NHn[Gen]	red-GEN	‘belonging to the color red’
<i>alın</i>	[Verb]	+ [Pos] + [Imp] + [A2sg]	offend-2SG.IMP	‘Get offended!’
<i>al</i>	[Verb]	+ [Pos] + [Imp] + YHn[A2pl]	take-2SG.HNR-IMP	‘(Please) take it!’
<i>al</i>	[Verb]	& Hn[Verb+Pass] + [Pos] + [Imp] + [A2sg]	take-PASS[2SG]	‘Get taken!’

With respect to syntactic properties, Turkish has a relatively free word order, which is constrained by discourse elements and information structure [19–26]. Even though SOV is the base word order, other permutations are highly utilized, as exemplified in Example 3.¹ The percentages were determined by Slobin and Bever [27] from 500 utterances of spontaneous speech.

- (3) a. Fatma Ahmet-i gör-dü. (SOV 48%)
 Fatma Ahmet-ACC see-PST
 ‘Fatma saw Ahmet.’
- b. Ahmet’i Fatma gördü. (OSV 8%)
- c. Fatma gördü Ahmet’i. (SVO 25%)
- d. Ahmet’i gördü Fatma. (OVS 13%)
- e. Gördü Fatma Ahmet’i. (VSO 6%)
- f. Gördü Ahmet’i Fatma. (VOS <1%)

Adapted from [20]

¹Conventions used in this thesis are as follows: 1 = first person, 2 = second person, 3 = third person, ABL = ablative, ACC = accusative, AOR = aorist, CAUS = causative, CL = classifier, COM = comitative, COND = conditional, COP = copula, CVB = converb, DAT = dative, EMPH = emphasis, FUT = future, GEN = genitive, HNR = honorific, IMP = imperative, LOC = locative, NEG = negative, NMLZ = nominalizer, PASS = passive, PL = plural, POSS = possessive, PROG = progressive, PST = past, Q = question particle, SG = singular. The dash symbol (-) in linguistics examples marks morpheme boundary, the equal sign (=) is used when the morpheme attached to a base is a clitic. The tilde ~ is used to indicate partial replication. The asterisk * at the beginning of a sentence shows the sentence’s ungrammaticality, and the percentage symbol (%) shows the marginal acceptability of the sentence. Additionally, we presented the analytic words within a box when they are segmented for annotation.

As for the case system, every argument in a sentence needs to host a case according to its syntactic role, semantic contribution, or the lexical selection of the phrasal head [28]. These groupings, however, are not clear cut and there is not always a one-to-one correspondence between cases and their roles.

Moreover, Turkish is a pro-drop language in which the subject can be elided when it is retrievable from the given discourse [29, 30]. Overt subjects are used only to convey certain discourse and/or pragmatic effects, such as a change in context or focus. However, the subject is also retrievable from the agreement marker on the verb. In addition to these properties, Turkish is also a null object language, even though the language does not have an overt agreement marker available for this process [31]. If the object of a sentence is retrievable from the given discourse, speakers may omit the object without any overt marking on the verb. The final issue with Turkish syntax lies in the fact that it frequently makes use of nominalization processes for embedded clauses [32]. With certain nominalizer suffixes, the embedded sentences may function as an adverbial, an adjectival, or a nominal [11].

2.2. Code-Switching

Code-switching (CS) is the process of generating utterances by combining phrases and word forms from multiple languages. This is a phenomenon observed frequently in utterances of bilingual speakers [33]. Figure 2.1 shows an example to this type of utterance formation.

Within a sentence, code-switching can occur *intra-sentential* or *intra-word*. Intra-sentential CS means that words, phrases, or clauses from more than one language are used together in a sentence. Intra-word CS is the mixing of two or more languages in a single word. The example in Figure 2.1 has both intra-sentential and intra-word CS.

There are a number of challenges in performing computational analysis of CS language pairs [34, 35]. Some of them are listed below:

- The source of code-switched data are usually spoken text or social media [36]. Hence, processing CS data requires solving additional problems that arise from the non-canonical nature of the data.
- CS language pairs are quite low-resource. Annotated data is usually inadequate for most of the NLP tasks, it is even absent for some of them. Besides, CS language pairs suffer from the shortage on resources required by deep neural models such as pretrained embeddings, language models, or even raw data.
- Code-switched texts include words from at least two different languages and each language composing the CS language pair inherits its own structural difficulties. The challenge amplifies as the linguistic difference between the composing languages increases.
- In the presence of intra-word code-switching in the data, out-of-vocabulary problem arises as the probability of the created CS word being an unknown word becomes very likely.

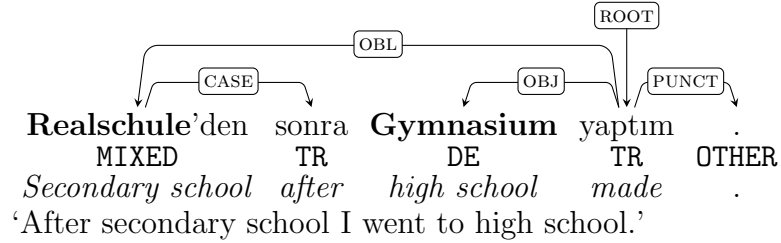


Figure 2.1. Dependency tree of a code-switched sentence from the Turkish-German SAGT Treebank. Language ID of each token is located below the token. TR stands for *Turkish*, DE for *German*, MIXED for tokens with intra-word code-switching, OTHER is for punctuation. German tokens and token parts are shown in bold.

3. THEORETICAL BACKGROUND

3.1. Dependency Parsing Approaches

There have been different approaches for dependency parsing. They are classified mainly into two categories, namely grammar-driven methods and data-driven methods.

3.1.1. Grammar-driven Methods

Grammar-driven dependency parsing relies on a formalization of a dependency grammar. These parsing methods usually have complex grammars and parsing strategies. According to Nivre [37], they can be divided into two categories.

3.1.1.1. Context-free-based Methods. These grammar-driven dependency parsing methods use a lexicalized context-free grammar, and so can use standard context-free parsing algorithms for dependency parsing. However, these methods are restricted to generate projective and unlabeled dependency trees only.

3.1.1.2. Constraint-based Methods. These methods consider parsing as a constraint satisfaction problem and define the grammar using constraints. The parsing problem is NP-complete, but can approximately be solved using eliminative parsing methods.

3.1.2. Data-driven Methods

Due to the availability of large amount of annotated data in recent years, data-driven dependency parsing has become more popular for parsing natural language texts. These approaches have two main subcategories.

3.1.2.1. Transition-based Methods. Transition-based methods induce a model to predict the next transition in a state machine, given the parse history. The model chooses the highest-scored transition at every step until the transition sequence is completed [38]. Generally, a transition-based model starts with an empty stack and a buffer of all input words. Then, at every step either a word is shifted from buffer to stack or an arc is defined between two words in the stack. This procedure continues until the buffer is empty. These parse actions are decided by a learning model.

3.1.2.2. Graph-based Methods. Graph-based methods first create a set of candidate dependency graphs for the target sentence. Then a model is induced to score these candidate graphs and finally the highest-scored dependency graph is searched to be the dependency parse of the sentence [38].

In graph-based dependency parsing approaches, dependency trees of sentences are represented using directed connected graphs. Formally, the directed connected graph G_S of a sentence S is defined as follows:

$$G_S = (V_S, E_S, W) \quad \text{such that} \quad V_S = \{S_0 = ROOT, S_1, \dots, S_n\} \quad \text{and} \\ E_S = \{(i, j) : i \neq j, (i, j) \in [0 : n] \times [1 : n]\},$$

where V_S is the set of nodes representing words in S , E_S is the set of directed edges between the nodes, and W is a function that assigns a weight to every directed edge in E_S . $W(i, j)$ denotes the probability of word j being dependent of word i [39].

Given a sentence, graph-based parsing approaches aim to find the highest scoring dependency tree of the sentence, which is equivalent to finding the Maximum Spanning Tree (MST) of the directed connected graph of the sentence.

3.2. Related Deep Learning Methodologies

Deep learning recently has become one of the most preferred data-driven approaches to dependency parsing task. In the following sections, we give information about the deep learning techniques used in this thesis.

3.2.1. Long Short-Term Memory Networks

Long short-term memory (LSTM) networks are a kind of recurrent neural networks (RNN) which can learn long-term dependencies. LSTMs are one of the most frequently used deep learning methods in dependency parsing. Different versions of this method are utilized and have shown good performance in dependency parsing [40–42].

An LSTM has one input layer, one recurrent LSTM layer, and one output layer. Input layer is connected to the LSTM layer. Each LSTM cell consists of three types of multiplicative gates. These are the input gate, the output gate, and the forget gate. The task of the input gate is to learn new information from the input to this LSTM cell. The forget gate decides whether the information that comes from the previous time step is to be remembered or unimportant and therefore can be omitted. The output gate passes the updated information to the next time step. Formally, an LSTM network maps an input sequence x to an output sequence y for each time step t as follows:

$$\begin{aligned} i_t &= \sigma(W_{ix}x_t + W_{ih}h_{t-1} + W_{ic}c_{t-1} + b_i) \\ f_t &= \sigma(W_{fx}x_t + W_{fh}h_{t-1} + W_{fc}c_{t-1} + b_f) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_{cx}x_t + W_{ch}h_{t-1} + b_c) \\ o_t &= \sigma(W_{ox}x_t + W_{oh}h_{t-1} + W_{oc}c_t + b_o) \\ h_t &= o_t \odot \tanh(c_t) \\ y_t &= g(h_t), \end{aligned}$$

where W denotes the weight matrices, b vectors are the bias vectors, i , f , o , and c are the input gate, forget gate, output gate, and cell activation vectors. h_t is the hidden state and y_t is the output of the current time step t . σ and g are the logistic sigmoid function and an arbitrary differentiable function, respectively [40, 43].

3.2.1.1. Stack-LSTM. A modified version of the standard LSTM architecture is the stack-LSTM model [40] which adjusts LSTM’s strict left-to-right order in the processing of the input. Different from the standard implementation, the current location of the

stack pointer in a stack-LSTM controls which LSTM cell’s c_{t-1} and h_{t-1} values are used in new memory cell computations [40].

3.2.1.2. Bidirectional LSTMs. Bidirectional LSTM (BiLSTM) is another extension of the standard LSTM architecture. As its name suggests, BiLSTMs handle the input sequence in both directions, i.e., from past to future (forward) and future to past (backwards). BiLSTMs have two separate hidden layers for each direction which are then combined and fed to the same output layer. This way, additional context can be learnt by the network [42].

3.2.2. Transformer-based Language Models

3.2.2.1. BERT. BERT (Bidirectional Encoder Representations from Transformers) is an encoder-based language representation model presented by Devlin *et al.* [44]. The architecture of a BERT model is a multilayer bidirectional Transformer encoder [45] and BERT uses WordPiece embeddings [46] for input representation. The two steps in training BERT includes pre-training and fine-tuning. In the pre-training phase, the BERT model is trained on unlabeled data over two unsupervised tasks: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). In MLM, the task is predicting the masked input tokens that are randomly chosen. In the NSP scheme, the model is given a pair of sentences as input and learns to predict if the second sentence is originally the subsequent sentence of the first sentence. In fine-tuning, the pretrained parameters of the model are fine-tuned using labeled data from a downstream task.

3.2.2.2. XLM-R. XLM-R stands for Cross Lingual Model - Roberta. Like BERT, it uses the Transformer model in [45] and is trained only using the MLM objective. Input is represented with Sentence Piece subword tokenizer [47]. The main difference between XLM-R and BERT relies in the size of the pretraining data. While XLM-R is pretrained on 2.5TB of unlabeled text data across 100 languages extracted from CommonCrawl Corpus [48], the multilingual version of BERT is pretrained on a much

smaller Wikipedia Corpus for 104 languages. Another difference of XLM-R from the BERT model is the better representation of low-resource languages in its pretraining data which makes XLM-R a frequently preferred model for such languages [49].

3.3. Evaluation Metrics

In order to evaluate our dependency parsers, we use the attachment score metric. Attachment score is the most commonly used evaluation metric in dependency parsing. There are different types of attachment scores. The metrics we used in the experiments throughout the thesis are explained below.

Unlabeled Attachment Score (UAS). UAS metric is defined as the percentage of words that are attached to the correct head.

Labeled Attachment Score (LAS). LAS is defined as the percentage of words that are attached to the correct head with the correct dependency type.

Morphology-aware Labeled Attachment Score (MLAS). MLAS is similar to the LAS metric, however it mainly focuses on dependencies between content words and treats function words as features of content words. It also takes the POS tags and morphological features into account [50].

Bi-lexical Dependency Score (BLEX). BLEX is similar to MLAS in focusing on relations between content words. However it includes lemmatization instead of morphological features to the evaluation [50].

All metrics are defined as F1 scores. We compute precision (P) as the number of correct dependency relations divided by the number of system-produced word nodes and recall (R) as the number of correct relations divided by the number of gold-standard word nodes. Then we define each metric listed above by calculating the F1 measure:

$$F1 \text{ score} = \frac{2PR}{P + R}.$$

4. THE BOUN TREEBANK: A NEW AND HIGH QUALITY CORPUS FOR TURKISH DEPENDENCY PARSING

4.1. Introduction

The field of natural language processing has seen an influx of various treebanks following the introduction of the treebanks in [51], [52], and [53]. These treebanks paved the way for today’s ever-growing NLP framework, consisting of NLP applications, treebanks, and tools. Even though the value of a treebank cannot be judged solely by its number of sentences, previous research has shown that the size of a treebank may affect its utility in downstream NLP tasks [54]. Among the many languages with a growing treebank inventory, Turkish has been one of the less fortunate languages. As of the UD version 2.9 [3], the largest UD treebank is the UD German-HDT Treebank which consists of 190,000 sentences [55], yet the most frequently used Turkish treebank IMST includes only 5,635 sentences. Turkish has posed an enormous challenge for NLP studies due to its complex network of inflectional and derivational morphology, as well as its highly flexible word order. One of the first attempts to create a structured treebank for Turkish was initiated in the studies of Atalay *et al.* [56] and Oflazer *et al.* [57]. Following these studies, many more Turkish treebanking efforts were introduced [58–60]. However, most of these efforts contained a small volume of Turkish sentences, and some of them were re-introduced versions of already existing treebanks in a different annotation scheme.

In this chapter, we aim to contribute to the limited NLP resources in Turkish by annotating a part of a brand new corpus that has not been approached with a syntactic perspective before, namely the Turkish National Corpus (TNC) [61]. TNC is an online corpus that contains 50 million words. The BOUN Treebank, which is introduced in this chapter, includes 9,761 sentences extracted from five different text types in TNC, i.e. essays, broadsheet national newspapers, instructional texts, popular culture

articles, and biographical texts. These sentences have not been introduced within a treebank previously. We manually annotated the syntactic dependency relations of the sentences following the up-to-date UD annotation scheme.

Through a discussion of the annotation decisions made in the creation of the BOUN Treebank, we present our take on the annotation of Turkish data, including the challenges that the copular clitic, embedded constructions, compounds, and lexical cases pose. Turkish treebanking studies present an inconsistent picture in the annotation of such constructions, even though these linguistic phenomena are observed and studied extensively within Turkish linguistic studies.

In addition, we report the results of the dependency parsing experiments we made on the newly introduced BOUN Treebank together with previous Turkish treebanks. The results show that increasing the size of the training set has a positive effect on the parsing success for Turkish. We observe that using the UD annotation scheme more faithfully and in a unified manner within Turkish UD treebanks offers an increase in the UAS and LAS scores. We also report individual parsing scores for different text types within our new treebank.

This chapter is based on the collaborative work with the Department of Linguistics at Boğaziçi University published as [11] and supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) under grant number 117E971 and as BİDEB 2211 graduate scholarship.

4.2. Related Work

The initial groundwork for Turkish treebanks was laid in [56] and [57] following the studies on treebanks for languages such as English, German, Dutch, and many more [51–53, 62, 63]. The first of its kind, the METU-Sabancı Treebank (MST) consists of 5,635 sentences, a subset of the METU corpus that reportedly includes 16 different text types such as newspaper articles and novels [64]. Oflazer *et al.* [57] encoded

both morphological complexities and syntactic relations. Due to the productive use of derivational suffixes, they explicitly spelled out every inflection and derivation within a word. As for the syntactic representation, Atalay *et al.* [56] used a dependency grammar in order to bypass the problem of constituency in Turkish, which arises from the relatively free word order of the language.

Branching off the works in [56] and [57], a small treebank with the name of ITU Validation set for MST was introduced. It contains 300 sentences from 3 different genres. The treebank was introduced as a test set for MST in the CoNLL 2007 Shared Task [65]. The treebank was annotated by two annotators using a cross-checking process. Following this work, MST was re-annotated by Sulubacak *et al.* [2] from ground up with revisions made in syntactic relations and morphological parsing. The latest version was renamed as the ITU-METU-Sabancı Treebank (IMST). Due to certain limitations, Sulubacak *et al.* [2] employed only one linguist and several NLP specialists. The annotation process was arranged in such a way that there was no cross-checking between the works of the annotators. Moreover, inter-annotator agreement scores, details regarding the decision process among annotators, and the adjudication process have not been reported. Nevertheless, this re-annotation solved many issues regarding MST by proposing a new annotation scheme. Even though problems such as semantic incoherence in the usage of annotation tags and ambiguous annotation were resolved to a great extent, the non-communicative nature of the annotation process led to a handful of inconsistencies.

The inconsistencies in IMST were also carried over to its latest version, which utilizes automatic conversions of the tags from IMST to the UD framework [60]. Mappings of syntactic and morphological representations were also included. Consequently, IMST was made more explanatory and clear thanks to the systematically added additional dependencies. While its previous version had 16 dependency relations, 47 morphological features, and 11 part of speech types, the latest version of IMST upped these numbers to 29, 66, and 14, respectively. Yet, the erroneous dependency tagging resulting from morpho-phonological syncretisms lingered long after the publication of

the treebank. Moreover, no post-editing effort has been reported. There have been four updates since the first release of the IMST Treebank in UD scheme, but there are still mistakes that can be corrected through a post-editing process, such as the punctuation marks tagged as roots, reversed head-dependent relations, and typos in the names of syntactic relations.

Apart from the treebanks originating from MST, many other treebanks have emerged. Some of these treebanks can be grouped under the class of *parallel treebanks*. The first of these parallel treebanks is the Swedish-Turkish Parallel Treebank (STPT). Megyesi *et al.* [66] published their parallel treebank containing 145,000 tokens in Turkish and 160,000 tokens in Swedish. Following this work, Megyesi *et al.* [58] published the Swedish-Turkish-English Parallel Treebank (STEPT). This treebank includes 300,000 tokens in Swedish, 160,000 tokens in Turkish, and 150,000 tokens in English. All the treebanks utilized the same morphological and syntactical parsing tools. For Swedish morphology, the Trigrams‘n’Tags tagger [67] trained on Swedish [68] was used. On the other hand, Turkish data were first analyzed using the morphological parser in [69], and its accuracy was enhanced through the morphological disambiguator proposed in [70]. The Turkish and Swedish treebanks were annotated using the MaltParser [71] that was trained with the Swedish treebank Talbanken05 [72] and MST [57], respectively.

Another parallel treebank introduced for Turkish is the Turkish PUD Treebank, which adopts the UD framework. The Turkish PUD Treebank was published as part of a collaborative effort, the CoNLL 2017 Shared Task on Multilingual Parsing from Raw Text to Universal Dependencies [73]. Sentences for this collaborative treebank were drawn from newspapers and Wikipedia. The same 1,000 sentences were translated into more than 40 languages and manually annotated in line with the universal annotation guidelines of Google. After the annotation, the Turkish PUD Treebank was automatically converted to the UD style.

Moreover, there are three treebanks that consist of informal texts. One such treebank was introduced by Pamay *et al.* [74] under the name of ITU Web Treebank

(IWT). In IWT, non-canonical data were included such as the usage of punctuations in emoticons, abbreviated writing such as *kib* that stands for *kendine iyi bak* (take care of yourself), and non-standard writing conventions as in *saol* instead of *sağol* (thanks). Later on, the UD version of IWT was also introduced [75]. Another web treebank has recently been presented by Kayadelen *et al.* [76], which is larger than the previous Turkish treebanks in terms of word count, but still smaller than the BOUN Treebank that we introduce in this thesis. Kayadelen *et al.* [76] used a set of dependency labels similar to the UD framework. However, they diverge from the UD framework in certain issues such as postpositions, indirect objects, and oblique arguments. The Turkish-German Code-Switching Treebank [77] is another treebank, in which they did not use formal texts. The Turkish-German Code-Switching Treebank consists of bilingual conversation transcriptions as well as their morphological and syntactic annotation. This treebank includes 48 unique conversations and 2,184 Turkish-German bilingual sentences that have been annotated with respect to the language in use.

There is also one grammar book-based treebank introduced in [78]. The Grammar Book Treebank (GB) is the first UD attempt in Turkish treebanking. In this treebank, data were collected from a reference grammar book for Turkish written by Göksel and Kerslake [32]. It includes 2,803 items that are either sentences or sentence fragments from the grammar book. It utilized TRMorph [79] for morphological analyses and the proper morphological annotations were manually selected amongst the suggestions proposed by TRMorph. The sentences were manually annotated in the native UD-style.

In addition to these treebank initiatives, in one of our previous works, we manually corrected the syntactic annotations in the Turkish PUD and IMST treebanks [80, 81] with the aim of unifying syntactic annotation scheme in Turkish treebanking. In these works, we selected the treebanks that were not annotated natively in the UD style and unified the annotation scheme. This process improved the UAS score for the IMST Treebank from 72.49 to 75.49 and caused only a 0.9 point decrease in the LAS score (from 66.43 to 65.53) in our experiments with the Stanford’s neural dependency parser [82], despite the number of unique dependency tags increasing from 31 to 40 with

the newly included dependency types [80]. On the other hand, there was a decrease in the parsing accuracy for the re-annotated version of the PUD Treebank in terms of the attachment scores. While the parser achieved an UAS score of 79.52 and a LAS score of 73.81 on the previous version of the PUD Treebank, its attachment scores for the re-annotated version were 78.70 UAS and 70.01 LAS [81]. We want to note that, we used 5-fold cross validation for the evaluation of the PUD Treebank due to its small size. In each fold, the parser had only 600 sentences for training, and 200 sentences were used as the development set. The evaluation was done on the remaining 200 sentences. The small size of the PUD Treebank, which was originally used only for evaluation purposes (not for training) in the CoNLL 2017 Shared Task [73], renders the results less reliable. Following these studies, with the annotation scheme we unified, we manually annotated the BOUN Treebank, which we present in this thesis. In Table 4.1, we present basic statistics about the BOUN Treebank and compare it to the previous monolingual Turkish treebanks. If both UD and non-UD versions are available for a treebank, we only included the UD version in the table.²

Table 4.1. Comparison of the BOUN Treebank to previous monolingual Turkish treebanks.

	IMST	IWT-UD	GB	PUD	BOUN
Num. of sentences	5,635	5,009	2,880	1,000	9,761
Num. of tokens	56,396	44,463	16,803	16,536	121,214
Avg. token count per sentence	10.01	8.88	5.83	16.53	12.41
Avg. dependency arc length	2.71	2.13	1.77	2.91	2.86
Num. of unique POS tags	14	15	16	16	17
Num. of unique features	66	54	79	59	56
Num. of unique dependencies	32	28	41	40	41

²UD version number of these treebanks is 2.7. Turkish PUD version 2.7 is our re-annotated version.

4.3. The BOUN Treebank

We introduce a treebank [83] that consists of 9,761 sentences which form a subset of the Turkish National Corpus (TNC) [61]. TNC includes 50 million words from various text types, and encompasses sentences from a 20 year period between 1990 and 2009. The principles of the British National Corpus were followed in terms of the selection of the domains. Table 4.2 shows the percentages of different domains and media used in TNC.

Table 4.2. Composition of the written component of TNC using words as the measurement unit, adapted from [61].

Domain	%	Medium	%
Imaginative	19	Books	58
Social Science	16	Periodicals	32
Art	7	Miscellaneous published	5
Commence/Finance	8	Miscellaneous unpublished	3
Belief and Thought	4	Written-to-be-spoken	2
World Affairs	20		
Applied Science	8		
Nature Science	4		
Leisure	14		

In our treebank, we included the following text types: essays, broadsheet national newspapers, instructional texts, popular culture articles, and biographical texts. Approximately 2,000 sentences were randomly selected from each of these registers. All of the selected sentences were written items and were not from the spoken medium. Our motivation for using these registers was to cover as many domains as possible using as few registers as possible, while not compromising variations in length, formality, and literary quality. The basic statistics for the BOUN Treebank and its different sections are provided in Table 4.3.

Table 4.3. Sentence and word statistics for the different sections of the BOUN Treebank. The difference between the number of tokens and words is due to multi-word expressions being represented with a single token, but with multiple words.

Treebank	Num. of sentences	Num.of tokens	Num. of word forms
Essays	1,953	27,007	27,557
Broadsheet National Newspapers	1,898	29,307	29,386
Instructional Texts	1,976	20,442	20,625
Popular Culture Articles	1,962	21,067	21,263
Biographical Texts	1,972	23,391	23,553
Total	9,761	121,214	122,384

Before the manual annotation of the BOUN Treebank, the sentences were first automatically annotated using an end-to-end parsing pipeline tool that parses raw texts to UD dependencies in CoNNL-U format with POS and morphological tagging information [84]. The manual syntactic annotation of sentences were then performed on this automatically generated CoNNL-U versions of the corpus sentences. In the manual annotation process, we followed the UD syntactic relation tags. Before the annotation process started, we first reviewed the dependency relations in use within the UD framework. Upon reviewing the definitions, we created and annotated a list of unique sentences that we believe are representative of the UD dependency relations in Turkish. Later on, we compared our sentences for certain dependency relations with the examples from already existing Turkish UD treebanks. If our examples and the UD examples were not parallel, we first discussed whether or not our interpretation was correct. We then discussed whether or not there should be any inclusions to the UD guidelines. These discussion were also brought up within the UD community.

After settling on the definitions of dependency relations, two Turkish native speaker linguists manually annotated the BOUN Treebank using our annotation tool presented in Section 8.1. Following the annotation process, two other linguists who did not participate in the manual annotation process cross-checked syntactic annotations of the two linguists. When a problematic sentence or an inconsistency was encountered,

discussions on the sentence and related sentences were held among the team members. After a decision was made, the necessary changes were applied uniformly.

In addition to the cross-checking process, we performed a partial double annotation in order to have a consistent annotation scheme before the annotation process of the BOUN Treebank started. For this purpose, the annotators performed an additional annotation task independently for the same set of 1,000 randomly selected sentences. The disagreements were discussed and resolved with the entire team of linguists and NLP specialists. The Cohen’s Kappa measure of inter-annotator agreement for finding the correct dependency label of the relations is found to be 0.82. The unlabeled and labeled attachment scores between the annotations are 0.83 and 0.75, respectively.

4.3.1. Levels of Annotation

4.3.1.1. Morphology. Turkish makes use of affixation much more frequently than any other word-formation process. Even though it adds an immense complexity to its word level representation, patterns within the Turkish word-formation process allowed previous research to formulate morphological disambiguators that dissect word-level dependencies. One such work was introduced by Sak *et al.* [85]. Their morphological parser is able to run independently of any other external system and is capable of providing the correct morphological analysis with 98% accuracy using contextual cues, such as the two previous tags.

In the morphological annotation of the BOUN Treebank, we decided to use the morphological analyzer and disambiguator of Sak *et al.* [85]. For this purpose, the tokenized sentences were first given to the morphological parser. The output of the parser was converted to the corresponding UD features automatically. In rare cases where the morphological parser did not return a morphological analysis for a token, the morphological features column from the Turku pipeline [84] for this token was used. The same operation was done for the lemmas of the tokens as well.

Our preference for the morphological tagger of Sak *et al.* [85] instead of the morphological tagger of the Turku parsing pipeline [84], which we used for the automatic processing of the treebank in the first step, is due to their comparison in terms of the token-based accuracy, and the feature-based recall, precision, and f-measure metrics. After randomly selecting 50 words from every text type in the BOUN Treebank (a total of 250 unique tokens excluding punctuations for the five text types), we encoded the errors made by the morphological parsers. The results are shown in Table 4.4. *Token Accuracy* column represents the token-based accuracy, namely the percentage of words for which correct morphological analyses are produced. *Recall* column represents the ratio of the number of correct morphological features to the number of morphological features in the gold standard. *Precision* column encodes the ratio of the number of correct morphological features to the total number of morphological features predicted by the morphological parser. The *F1-measure* column is the harmonic mean of precision and recall. Our scores align with the scores reported in the original study of Sak *et al.* [85], even though their test set and our set here consist of different text types. While they only used newspaper corpora in the test set, we tested the parser using different text types including broadsheet national newspapers, essays, instructional texts, biographical texts, and popular culture articles.

Table 4.4. The performance of morphological taggers of Sak *et al.* [85] and Turku pipeline [84] on BOUN Treebank.

Morphological Tagger	Token Accuracy	Recall	Precision	F1-measure
Sak <i>et al.</i> [85]	0.91	0.94	0.95	0.94
Turku pipeline	0.82	0.89	0.83	0.86

The morphological parser of Sak *et al.* [85] does not provide morphological tags in UD format. So, we automatically converted its output to the UD format. In this process, we maximally used the morphological features from the UD framework. When there is no clear-cut mapping between the features that we acquired from the morphological parser in [85] and features proposed in the UD framework, we used the features

previously suggested in the works of Çöltekin [86], Tyers *et al.* [87], and Sulubacak and Eryiğit [75]. These features were already stated in the UD guidelines. Table 4.5 shows the automatic conversion from the results of the morphological disambiguator of Sak *et al.* [85]

Table 4.5. Mappings of morphological features from the notation of Sak *et al.* [85] to the features used in the UD framework.

Sak <i>et al.</i> [85]	UD	Sak <i>et al.</i> [85]	UD
A1sg	Number=Sing Person=1	ByDoingSo	VerbForm=Conv Mood=Imp
A2sg	Number=Sing Person=2	Pos	Polarity=Pos
A3sg	Number=Sing Person=3	Neg	Polarity=Neg
A1pl	Number=Plur Person=1	Past	Aspect=Perf Tense=Past Evident=Fh
A2pl	Number=Plur Person=2	Narr	Tense=Past Evident=Nfh
A3pl	Number=Plur Person=3	Fut	Tense=Fut Aspect=Imp
P1sg	Number[psor]=Sing Person[psor]=1	Aor	Tense=Aor Aspect=Hab
P2sg	Number[psor]=Sing Person[psor]=2	Pres	Tense=Pres Aspect=Imp
P3sg	Number[psor]=Sing Person[psor]=3	Desr	Mood=Des
P1pl	Number[psor]=Plur Person[psor]=1	Cond	Mood=Cnd
P2pl	Number[psor]=Plur Person[psor]=2	Neces	Mood=Nec
P3pl	Number[psor]=Plur Person[psor]=3	Opt	Mood=Opt
Abl	Case=Abl	Imp	Mood=Imp
Acc	Case=Acc	Prog1	Aspect=Prog Tense=Pres
Dat	Case=Dat	Prog2	Aspect=Prog Tense=Pres
Equ	Case=Equ	DemonsP	PronType=Dem
Gen	Case=Gen	QuesP	PronType=Ind
Ins	Case=Ins	ReflexP	PronType=Prs Reflex=Yes
Loc	Case=Loc	PersP	PronType=Prs
Nom	Case=Nom	QuantP	PronType=Ind
Pass	Voice=Pass	Card	NumType=Card
Caus	Voice=Cau	Ord	NumType=Ord
Reflex	Voice=Rfl	Distrib	NumType=Dist
Recip	Voice=Rcp	Ratio	NumType=Frac
Able	Mood=Abil	Range	NumType=Range
Repeat	Mood=Iter	Inf	VerbForm=Vnoun
Hastily	Mood=Rapid	FutPart	VerbForm=Part Tense=Future Aspect=Imp
Almost	Mood=Pro	PastPart	VerbForm=Part Tense=Past Aspect=Perf
Stay	Mood=Dur	PresPart	VerbForm=Part Tense=Pres
While	VerbForm=Conv Mood=Imp		

As it is clear from the table, depth of the morphological representation in [85] and that in the UD framework do not align perfectly, and there is no one-to-one mapping. For example, an output from [85] may include both **Narr** and **Past** features. In the automatic conversion, we would end up with **Tense=Past** twice and conflicting values for **Evident** feature. To resolve cases similar to these, we made use of simple rules that detect conflicting features due to our conversion and return appropriate features. Moreover, we used the morphological cues provided by the morphological parser to decide on the UPOS and lemma. All elements of our conversion and post-processing can be found at [88].

Table 4.6. An example sentence from our treebank encoded in CoNNL-U format.

# sent_id = ins_167									
# text = Sözü uzatıp seni merakta bıraktım galiba.									
# trans = Probably, I beat around the bush and kept you in suspense.									
<i>ID</i>	<i>FORM</i>	<i>LEMMA</i>	<i>UPOS</i>	<i>XPOS</i>	<i>FEATS</i>	<i>HEAD</i>	<i>DEPREL</i>	<i>DEPS</i>	<i>MISC</i>
1	Sözü	söz	NOUN	Noun	Case=Acc Number=Sing Person=3	2	obj	-	-
2	uzatıp	uza	VERB	Verb	Polarity=Pos VerbForm=Conv Voice=Cau	5	advcl	-	-
3	seni	sen	PRON	Pers	Case=Acc Number=Sing Person=2	5	obj	-	-
4	merakta	merak	NOUN	Noun	Case=Loc Number=Sing Person=3	5	obl	-	-
					Aspect=Perf Evident=Fh Number=Sing				
5	bıraktım	bırak	VERB	Verb	Person=1 Polarity=Pos VerbForm=Fin	0	root	-	-
					Tense=Past				
6	galiba	galiba	ADV	Adverb	-	5	advmod	-	SpaceAfter=No
7	.	.	PUNCT	Punc	-	5	punct	-	SpacesAfter=\n

In our treebank, in addition to the words, we encoded the lexical and grammatical properties of the words as sets of features and values for these features. We also encoded the lemma of every word separately, following the UD framework. Table 4.6 shows an example sentence encoded with the CoNNL-U format.

4.3.1.2. Syntax. In the BOUN Treebank, we decided to represent the relations among the parts of the sentences within a dependency framework. This decision has two main reasons. The main and the historical reason is the fact that the growth of Turkish treebanks has been mainly within the frameworks where the syntactic relations have

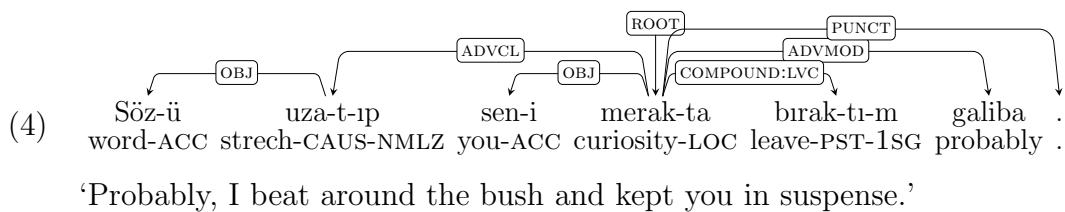
been represented with dependencies [69, 89]. The other reason is the fact that Turkish allows for phrases to be scrambled to pre-subject, post-verbal, and any clause-internal positions with specific constraints, which makes building constituency grammars quite difficult [90–93]. With these in mind, we wanted to stick with the conventional dependency framework and use the recently rising UD framework [94]. One of the main advantages of the UD framework is that it creates directly comparable sets of treebanks with regards to their syntactic representation due to its very nature.

Table 4.7. The dependency relation set of the BOUN Treebank.

Relation Type	Count	Percentage	Relation Type	Count	Percentage
<code>acl</code>	3,494	2.85%	<code>det</code>	4,938	4.03%
<code>advcl</code>	2,595	2.12%	<code>discourse</code>	381	0.31%
<code>advcl:cond</code>	269	0.22%	<code>dislocated</code>	28	0.02%
<code>advmod</code>	5,278	4.31%	<code>fixed</code>	12	0.01%
<code>advmod:emph</code>	1,724	1.41%	<code>flat</code>	2,039	1.67%
<code>amod</code>	7,869	6.43%	<code>goeswith</code>	4	0.002%
<code>appos</code>	506	0.41%	<code>iobj</code>	164	0.13%
<code>aux</code>	39	0.03%	<code>list</code>	40	0.03%
<code>aux:q</code>	269	0.22%	<code>mark</code>	117	0.10%
<code>case</code>	3,290	2.69%	<code>nmod</code>	1,371	1.12%
<code>cc</code>	2,800	2.29%	<code>nmod:poss</code>	10,393	8.49%
<code>cc:preconj</code>	134	0.11%	<code>nsubj</code>	8,499	6.94%
<code>ccomp</code>	1,512	1.24%	<code>nummod</code>	1,568	1.28%
<code>clf</code>	122	0.1%	<code>obj</code>	7,381	6.03%
<code>compound</code>	2,381	1.95%	<code>obl</code>	12,015	9.82%
<code>compound:lvc</code>	1,218	1.0%	<code>orphan</code>	84	0.07%
<code>compound:redup</code>	457	0.37%	<code>parataxis</code>	209	0.17%
<code>conj</code>	7,250	5.92%	<code>punct</code>	20,116	16.44%
<code>cop</code>	1,289	1.05%	<code>root</code>	9,761	7.97%
<code>csubj</code>	546	0.45%	<code>vocative</code>	88	0.07%
<code>dep</code>	9	0.01%	<code>xcomp</code>	125	0.01%

By following the UD framework, we implicitly encode two different syntactic information for each dependent: the category of the dependent and the function of this dependent with regards to its syntactic head. This is due to the grouping of the dependency relations introduced by the UD framework. The selection of the syntactic dependency relation for each dependent is mainly based on the functional category of the dependent in relation to the head and the structural category of the head. In terms of the functional category of the dependent, the UD framework differentiates the core arguments of clauses, non-core arguments of clauses, and dependents of nominal heads. As for the category of the dependent, the UD framework makes use of a taxonomy that distinguishes between function words, modifier words, nominals, and clausal elements. In addition to this classification, there are some other groupings which may be listed as coordination, multiword expressions, loose syntactic relation, sentential, and extra-sentential. Table 4.7 shows the dependency relations that we employed in the BOUN treebank with their counts and percentages.

Every dependency forms a relation between two segments within the sentence, building up to a non-binary and hierarchical representation of the sentence. In this way, nodes can have more than two child nodes and every node is accessible from the root node. This representation is shown in Example 4 using the sentence in Table 4.6.



4.3.2. Different Conventions Adopted in the Annotation Process

In the annotation process of the BOUN Treebank, we stayed faithful to the UD main tag set and the previous conventions of Turkish annotation schemes for the most part. However, there were some instances where we diverged from these conventions or made the linguistic reasoning behind them more explicit. In this section, we provide the justifications of our linguistic decisions for these instances. Our decisions are in the

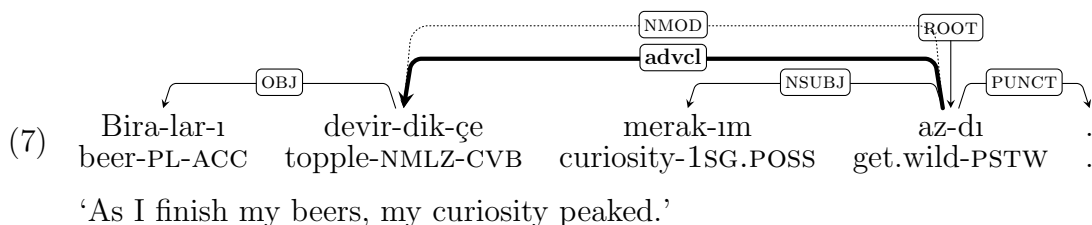
same spirit of unifying the annotation scheme within Turkish UD treebanks, which was done in our previous works [80,81]. Our main concern is to reflect linguistic adequacy in the BOUN Treebank following the Manning’s Law [95]. During all this work, we paid great attention to follow the previous discussion within the UD framework, such as the discussion on the copular clitic and the objecthood-case marking relation. In the following sections, we will first touch upon the issues where we believe the previous conventions in Turkish UD treebanking were erroneous according to UD. These issues include the annotation of the embedded sentences, the treatment of copular verb, the analysis of compounds, and the annotation of classifiers. Next, we will discuss the issue of objecthood and the case marking relation in Turkish, where we adopt a simpler analysis that has been used in other dependency grammars instead of the recently discussed UD alternatives.

4.3.2.1. Annotation of Embedded Clauses. The first issue where we diverged from the previous annotation conventions is the annotation of embedded clauses. In the previous Turkish treebanks, the annotation of embedded clauses did not reflect the inner hierarchy that a clause by definition possesses. This is mostly due to the morphological aspect of the most common embedding strategy in Turkish: nominalization. Due to nominalization, embedded clauses in Turkish can be regarded as nominals since they behave exactly like nominals: They can be marked with an accusative case, can be substituted with any other nominal, and can carry genitive-possessive cases as person marking. This phenomenon is shown in Example 5. The embedded clause in the given sentence is shown with square brackets. The whole square bracket can be replaced with a simple noun, like *otobüs* (bus), or a complex noun phrase like *senin otobüsün* (your bus) as in Example 6.

- (5) [Sen-in otobüs-ü sür-düğ-ün]-ü gör-dü-m.
 [you-GEN bus-ACC drive-NMLZ-POSS]-ACC see-PST-1SG
 ‘I saw that you drove the bus.’
- (6) Sen-in otobüs-ün-ü gör-dü-m.
 you-GEN bus-POSS-ACC see-PST-1SG
 ‘I saw your bus.’

Due to these surface level morphological and syntactic similarities, previous Turkish treebanks in the UD framework, with the exception of the Grammar Book Treebank [78], used dependency relation `obj` instead of `ccomp`, `nsubj` instead of `csubj`, `amod` instead of `acl`, and `advmod` instead of `advcl` to mark the relation of the embedded clause with the matrix verb. In our annotation process, we emphasized the clausal nature of these embedded sentences and their syntactic derivation by focusing on their internal structure reflecting the existence of a temporal domain in the embedded clause. For instance, Example 5 would be nonsensical if we had the time adverb *tomorrow* within the embedded clause. This ungrammaticality is due to the tense information introduced by the nominalizer ‘-düg’ in the example sentence. If there were an adverb like *tomorrow* in an embedded clause marked with ‘-düg’, the previous annotation scheme would not be able to detect the ungrammaticality. However, our annotation scheme is able to detect this ungrammaticality.

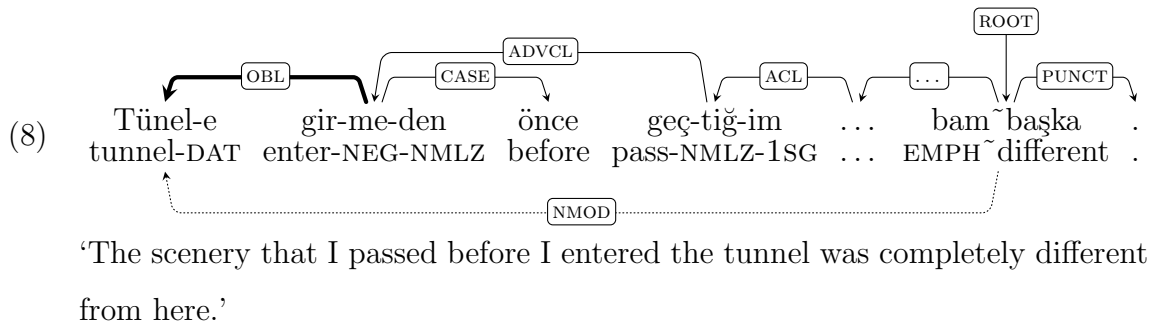
The same argumentation applies to converbs, as well. Converbs are verbal elements of a non-finite adverbial clause [32]. They may act as adverbial adjuncts or as discourse connectives. In the previous annotation processes of Turkish, they were annotated as `nmod`. The reason behind this annotation is again the fact that they behave like nominals; they may be marked with inflectional and derivational suffixes that normally nouns bear. Considering their clausal properties, such as their temporal domain, their ability to host a subject, an object, and a tense/aspect/modality information, we annotated them as `advcl` as in Example 7.³



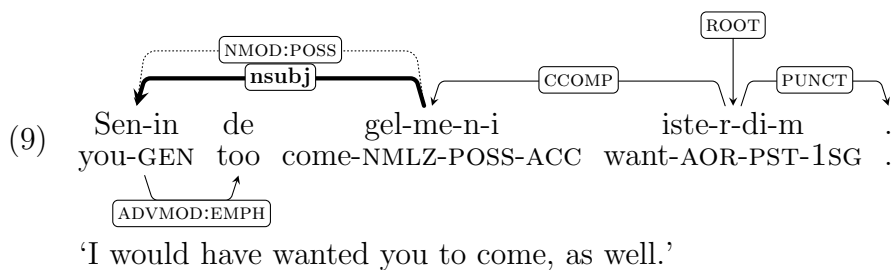
In addition to the annotation of the whole embedded clause, dependents within the embedded clause were erroneously annotated in the previous Turkish annotation schemes.

³Throughout this chapter, changes in the annotation convention introduced by us are shown with bold arcs, whereas the dashed arcs suggest previous annotations. The solid arcs represent unaltered dependencies. Every annotated tree that contains a bold arc in this thesis is taken from previous Turkish Treebanks, that is either the IMST Treebank or the Turkish PUD Treebank.

For example, an oblique of an embedded verb used to be attached to the root since the embedded verb is seen as a nominal, and not as a verb as in Example 8.



Likewise, the genitive subjects of embedded clauses were wrongly marked as a possessive nominal modifier, whereas they are one of the obligatory elements of the embedded structures. This wrong annotation in the previous treebanks is due to the fact that Turkish makes use of genitive-possessive structure for marking the agreement in an embedded clause as in Example 9 [32]. Despite the morphology, the word *senin* here serves as the subject. Example 10 shows the causativized version of the embedded verb in Example 9. When we causativize the subject of an intransitive verb, we expect the subject to be marked with an accusative case and act as a direct object. As seen in Examples 9 and 10, the word *sen* reflects the morphological reflex stemming from a syntactic voice change. Thus, it cannot be a modifier and it has to be an argument.



- (10) O-nun sen-i de getir-me-si-ni iste-r-di-m.
 he/she-GEN you-ACC too come.CAUS-NMLZ-POSS-ACC want-AOR-PST-1SG.
 ‘I would have wanted him/her to bring you, as well.’

Due to the reasons explained above, in the annotation of embedded clauses we used the dependency relations that emphasize the clausal nature of the nominalized verbs, i.e., *csubj*, *ccomp*, *advcl*, instead of the dependency relations that emphasize the final product of the local derivations, i.e., *nsubj*, *obj*, *advmod*, respectively.

4.3.2.2. Copular Clitic. One inconsistent issue within the Turkish treebanks was the annotation of the copular clitics. Copular clitics attached to the verbal bases and nominal bases were treated differently although they are essentially the same. While the copular clitics on verbal bases were not segmented, the copular clitics on nominal bases were segmented in previous Turkish treebanks. In this section, we will provide our analysis for the copular clitics where we segment all of them regardless of their bases.

The Turkish copular clitic is the grammaticalized version of the verb “be” which can be indicated as *i-*. This clitic *i-* has three allomorphs in Turkish: (i) analytic *i-*, (ii) suffixal *-y*, and (iii) zero-marked (\emptyset). The allomorphy of the analytic form is idiosyncratic, meaning the analytic copula form can be used in place of the suffixal copula forms most of the time. The analytic form can surface if suffixes *-di* (PST), *-se* (COND), and *-ken* (WHEN or WHILE) come atop a verb that already hosts a TAM (Tense/Aspect/Modality) marker. The analytic form can also surface in nominal sentences that are marked for tense other than the aorist (*-Ar/Ir*). However, the analytic form cannot surface with the suffix *-miş* (PRF), except for its use with the aorist as in *yapar imiş*, meaning *he/she used to do*. Examples 11a and 11b illustrate some examples of the analytic form.

- (11) a. analytic COP *i-*
-
- Okul-a var-acak i-di-m .
school-DAT reach-FUT be-PST-1SG .
'I was going to arrive to school.'
- b. analytic COP *i-*
-
- Okul-a var-acak i-ken .
school-DAT reach-FUT be-WHEN .
'I was about arriving to school. ...'

When both the base and the copular verb surface as a single syntactic word indicated with a box in the following examples, either *-y* (Examples 12a and 12b) or \emptyset (Examples 13a and 13b) is used.

- (12) a. zero-marked COP (\emptyset)
-
- Okul-a gel-ecek =ti-m .
school-DAT come-FUT =PST-1SG .
'I was going to come to school.'
- b. zero-marked COP (\emptyset)
-
- Okul-da öğretmen =di-m .
school-LOC teacher =PST-1SG .
'I was a teacher in the school.'

- (13) a. suffixal COP *-y*
-
- Okul-a gel-se =y-di-m .
 school-DAT come-COND =COP-PST-1SG .
 ‘If I went to school.’
- b. suffixal COP *-y*
-
- Okul-da öğrenci =y-se-m .
 school-LOC student =COP-COND-1SG .
 ‘If I was a student in the school...’

The selection between the \emptyset and *-y* is governed by the phonological characteristics of the previous sound; if the previous segment is a consonant \emptyset is used, otherwise *-y* is used. What is important for us is that the contribution of these copular clitics is the same for both nominal and verbal bases. In both cases, these copular clitics host the TAM information that cannot be carried by the base [96]. Hence, the TAM information itself also does not change according to the category of the stem.

Additionally, the stress patterns of the clitics that attach to nominal and verbal bases are identical. Most of the verbs and common nouns are stressed in the final syllable. When they are marked with a copular clitic, instead of the final syllable which is the copular clitic, the preceding syllable is stressed [96]. This property as well applies regardless of the base the clitic attaches to.

In addition to these characteristics, the copular clitic also has a clitic-like behaviour when it co-occurs with other clitics such as the question clitic *-mI*. In Example 14, the question clitic comes between the TAM marker and the copula.

- (14) Bu kitab-ı oku-yacak m₁=y-dı-n?
 this book-ACC read-FUT Q=COP-PST-2SG
 ‘Were you going to read this book?’

Another clue for the clitic status of the copula is its interaction with vowel harmony. When detached, it has its own phonological domain; thus vowel harmony processes do not percolate from the main verb to the copula as seen in Example 11a.

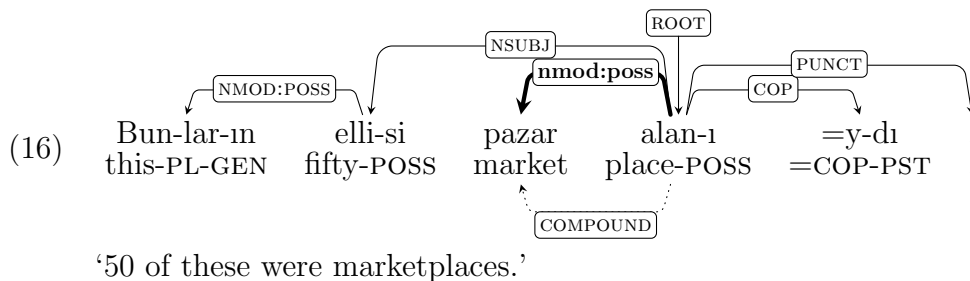
However, semantic contributions of TAM markers and their interaction with each other provides a counterpoint for segmenting the copular clitic. On a first look, verbs with a copular clitic seem to carry two different tense information. However, two consecutive TAM markers in Turkish do not imply two tenses. While one of them

still provides tense information, the other one implies additional aspect. Consider the verb *gelecektim* in Example 15. When either suffix (*-ecek* or *-ti*) is attached to a verb without any additional TAM marker, they mainly provide the tense information. When they are used together as in Example 15, the suffix *-ti* implies the tense information, and the suffix *-ecek* provides the prospective aspect information. This aspect of the copular clitic points towards a solution in which verbs with a copular clitic should be analyzed as a single unit.

- (15) Okul-a gel-ecek=ti-m ama fikri-m-i deđiřtir-di-m.
 school-DAT come-FUT=PST-1SG but mind-POSS.1SG-ACC change-PST-1SG
 ‘I was going to come to the school, but I changed my mind.’

After exchanging ideas on this issue within the UD community and considering points mentioned in this section, we decided to segment all instances of the copular verb *i-* as a copula (**cop**). With this change, we unified the treatment of all clitics that may attach to a root which include the question particle *=mı*, focus particles like *=da*, and copular verb particles; thus, we followed the UD dependency relations more faithfully.

4.3.2.3. Compound. Another inconsistent annotation in the previous Turkish treebanks was compounds and their classification. The UD framework suggests that **compound** should be tailored to each language with its particular morphosyntax. Mostly in Turkish PUD, also in other Turkish UD treebanks, constituents that carry a morphological marker for possessive-compounds are annotated as **compound** like in Example 16. The name ‘possessive-compounds’ is how the linguistic literature refers to it, but for our purposes we take it as a compositional structure and separate it from the UD dependency type ‘compound’. This means that our criteria for compound-hood are syntactic composition properties. We have modified cases with *-(s)I(n)* morphological marker as **nmod:poss**, which is already a convention in use in UD.



Turkish employs different strategies for compounding. These strategies can display differences in their morphological and phonological forms. For our purposes, we divide them into two: (i) compounds with the compound marker $-(s)I(n)$ and (ii) compounds without the compound marker $-(s)I(n)$. Some compound types without the compound marker are given in Example 17. These compounds are formed with different types of lexical inputs and can have varying degrees of morpho-phonological properties, none of which employs a compound marker. We annotated the compounds that do not employ a marker as **compound**.

- | | | | |
|------|---|---|--|
| (17) | a. Noun + Noun | c. Noun + Non-Word | in-di bin-di
on-PST off-PST |
| | şış kebab
skewer kebab
‘shish kebab’ | kitap mitap
book EMPH~book
‘book and whatnot’ | |
| | b. Non-word + Non-word | d. Adverb + Adverb | f. Adjective + Adjective |
| | abur cubur
... ..
‘junk food’ | bugün yarın
today tomorrow
‘soon’ | tive
kırık dökük
broken dowdy |
| | | e. Verb + Verb | ‘scrap’ |

The important distinction for our purposes is the existence of the compound marker $-(s)I(n)$. This marker is only observed in Noun+Noun compounds and most of these compounds can be turned into genitive-possessive constructions as in Example 18.

- | | | |
|------|---|--|
| (18) | a. Noun + Noun | b. Possessive construction |
| | okul bina-sı
school building-3SG
‘school building’ | okul-un bina-sı
school-GEN building-3SG
‘the school’s building’ |

We annotated Noun+Noun compounds that employ the compound marker $-(s)I(n)$ as **nmod:poss**. There are three reasons behind this decision. The first one is that the marker does not survive in possessive constructions, it is replaced by the possessive markers. If the possessor is 1SG or 2SG, the marker is replaced with first person singular possessive $-(I)m$ or the second person singular possessive $-(I)n$, respectively. If the possessor is 3SG the marker stays the same. The second reason is that any plural marking precedes the marker $-(s)I(n)$ as opposed to following it, just like in possessive

constructions (Example 19). The third reason is that compounds formed with the marker $-(s)I(n)$ can have their modifier be subject to questions, whereas compounds without it cannot (Example 20). Questions are considered to be extractions out of syntactic structures which cannot target parts of a word form.

- (19) a. ders kitap-(lar)-1 course book-PL-3SG ‘coursebook(s)’
 b. ders kitap-(lar)-1m course book-PL-1SG ‘my coursebook(s)’
 c. ders kitap-(lar)-1m course book-PL-2SG ‘your coursebook(s)’
 d. (o-nun) ders kitap-(lar)-1 (s/he-GEN) course book-PL-3SG ‘his/her coursebook(s)’
- (20) a. i. adana kebab A kebab ‘Adana kebab’
 ii. *Ne kebab ye-di? what kebab eat-PST[3SG] Intended ‘What type of kebab did (s/he) eat?’
 b. i. adana kebab-1 A kebab-3SG ‘Adana kebab’
 ii. Ne kebab-1 ye-di? what kebab-3SG eat-PST[3SG] ‘What type of kebab did (s/he) eat?’

As a result, (i) the marker $-(s)I(n)$ not surviving possessive constructions and the ability to transition from a compound to genitive-possessive construction shows that the marker $-(s)I(n)$ and possessive markers are in a disjunctive blocking relation. This suggests that they are competing for similar grammatical functions. (ii) The plural marker linearizes before the marker $-(s)I(n)$. If $-(s)I(n)$ was part of the word form, the plural marking should have linearized to the right of it. This shows that the marker $-(s)I(n)$ is not part of the word form. (iii) Parts of the construction formed by $-(s)I(n)$ can be targeted by questions. Question formations only target syntactic constituents and not part of word forms. This indicates that structures with $-(s)I(n)$ do not constitute an indivisible word form. All these three reasons make constructions involving $-(s)I(n)$ more syntactic (compositional) than morphological. This does not

unilaterally rule out the constructions with $-(s)I(n)$ as compounds, but within the framework of UD they are more suited to be classified as `nmod:poss` than `compound`.

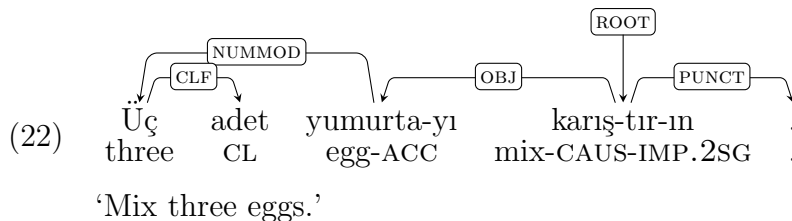
There is a robust linguistics discussion about the status of the marker $-(s)I(n)$ as being classified either as a compound or as an agreement marker. The word forms produced by it are actually referred to as ‘possessive compounds’ [97–100], introducing a dilemma even in its own name.

4.3.2.4. Classifier. The use of the classifier syntactic dependency (`clf`) was also inconsistent within the already existing Turkish UD treebanks. In the UD guidelines, the use of `clf` is limited to languages with highly grammaticized classifier systems. The difference between classifier languages and non-classifier languages is depicted with Chinese (classifier) and English (non-classifier). However, this distinction is not always clear-cut in other languages like Turkish [101]. According to Göksel and Kerslake [32], numerals can be followed by certain elements such as the enumerator *tane* (piece), measurement denoting words such as *dilim* (slice) and *şişe* (bottle), and membership/identity denoting words like *örnek* (example) and *kopya* (copy). They show that even though these elements are optional between a numeral and a noun, in partitive constructions with ablative cases, they are obligatorily used. The examples below show that the classifier *tane* (piece) is optional in sentences like Example 21a. However, when the classifier is in inflected form, deleting it makes the sentence ungrammatical as in Example 21b. The sentence becomes marginally acceptable when the inflection is concatenated to the numeral as in Example 21c.

- (21) a. Dört (tane) elma al-dı-m.
 four piece apple buy-PST-1SG
 ‘I bought four apples.’
- b. Küçük-ler-den on *(tane-si) yeter mi?
 small-PL-ABL ten piece-POSS enough Q
 ‘Will ten of the little ones be enough?’
- c. % Küçük-ler-den on-u yeter mi?
 small-PL-ABL ten-POSS enough Q
 ‘Will ten of the little ones be enough?’

Apart from the Turkish PUD Treebank, no previous Turkish treebank has used the `clf` syntactic dependency. In the Turkish PUD Treebank, both measure words and enumerators are annotated using `clf` dependency. As for the other Turkic treebanks, a measure word *bötelke* (bottle) in the Kazakh UD Treebank is annotated using `clf`. On the other hand, in the Uyghur UD Treebank, no `clf` is used. In addition to the UD Treebanks, other recent treebanks such as Kayadelen *et al.* [76] that use dependency grammar framework in their annotation, make use of the classifier dependency relation for both enumerators and measurement denoting words.

In the BOUN Treebank and our re-annotated versions of PUD and IMST, we annotated enumerators like *tane* (piece) and *adet* (piece) as classifiers and used the `clf` dependency relation. A slightly modified example sentence from our treebank can be seen in Example 22. One of the UD framework’s core ideas is to create a typologically comparable set of treebanks. In this direction, it is important to reflect the use of classifier words in Turkish, even if they are optional.



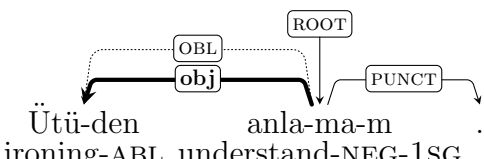
4.3.2.5. Core Arguments. Turkish also poses a problem with regards to the detection of core arguments. This problem stems from mainly two reasons: core arguments marked with a lexical case and object drop of the core arguments. Like Czech, Turkish allows its direct object to be marked with oblique cases. In addition to the structural accusative case, Turkish also makes use of dative, ablative, comitative and locative on objects, which are the cases that adjuncts can also take. Both the adjunct in Example 24 and the core argument in Example 23 are marked with the same case: COM (comitative). When there is no appropriate context that introduces the object earlier, a COM-marked NP becomes obligatory as in Example 23. However, Example 24 is completely fine regardless of the context and the existence of the COM-marked NP. This is because the COM-marked NP is a core argument in Example 23, whereas

it is an adjunct in Example 24. As it can be seen from the examples, Turkish can drop its object without any marking on the verb when it is available in the discourse or it is not contradictory within a given context. Since it is impossible to drop the new information or correction in the case of Example 23 without a context that introduces the direct object earlier, we conclude that the NP *kız kardeşiyle* (with her sister) is a core argument. If it were just an adjunct, the phrase could be omittable.

- (23) Serap *(kız kardeş-i-yle) hep dalga geç-er.
 Serap girl sibling-POSS-COM always make.fun-AOR.
 ‘Serap always makes fun of her sister.’

- (24) Serap okul-a (abla-sı-yla) gid-er.
 Serap school-DAT big.sister-POSS-COM go-AOR
 ‘Serap goes to school (with her elder sister).’

Oblique case marking of the core arguments together with the optionality of the contextually available core arguments yields a problem for the annotation process within a framework where the difference between core arguments and non-core arguments is a morphologically-apparent case marking as in the UD framework. Recent discussions in the UD framework also acknowledge this problem [102, 103]. They propose a new dependency relation: *obl:arg*. In our annotations, we used the *obj* dependency relation as in Example 25. The UD guidelines state that even though *obj* often carries an accusative case, it may surface with different case markers when the verb dictates a different form, in our case *lexical* cases like COM (Example 23) and ABL (Example 25). This approach is also utilized within the most recent Turkish treebank in which they did not distinguish between the objects with accusative case and the objects with non-accusative cases [76].

- (25) 
 ironing-ABL understand-NEG-1SG .
 ‘I do not know anything about ironing.’

Another core argument specified in the UD guidelines is the *iobj* argument. In their assessment of Turkic treebanks, Tyers *et al.* [87] suggest using case promotion or demotion in passivization or causativization as a clue for determining argumenthood.

When sentences are passivized in Turkish, the structural case accusative on the object is deleted in the transformation whereas oblique cases such as the ablative case is not deleted. They use this asymmetry to argue for a non-core analysis of oblique case marked objects. In their proposed annotation scheme, only tokens with non-oblique cases should be annotated as a core argument since only non-oblique cases go through case promotion or demotion. However, as we have previously shown in this section, objects marked with oblique cases behave the same as the objects marked with the accusative cases. Turkish can have oblique cases as a marker of objects even though they do not go through case demotion in passive sentences as in Example 26.

- (26) Ütü-den de anla-n-ma-z mı?
 ironing-ABL EMP understand-PASS-NEG-3SG Q?
 ‘How can one not know anything about ironing?’

Following the reasons specified in this section, we did not make use of case clues in the annotation of *iobj*, instead we utilized the effects born out of context. Following our annotation process, we should annotate the dative marked noun *bana* (to me) in Example 27 using the *iobj* dependency relation if we cannot omit it when the information is already available in the discourse. Without any existing prior context, one cannot omit the dative marked noun in sentences like Example 27 where the main predicate is ditransitive.

- (27) Deniz kitab-ı bana ver-di .
 Deniz book-ACC 1SG-DAT give-PST
 ‘Deniz gave me the book.’
-

In addition to our treebank, the *iobj* dependency relation is also used in other Turkish and Turkic treebanks. Prior to our re-annotation, the Turkish PUD Treebank already made use of this dependency relation. With our re-annotation, the IMST Treebank also utilizes the *iobj* dependency. The *iobj* relation is also used in a Turkic treebank: the UD Kazakh Treebank [87, 104]. We believe that the non-optionality of cases like *bana* (to me) in Example 27 and its already existing use in other Turkish and Turkic treebanks justify our usage as well.

4.3.2.6. Summary of the linguistic considerations. The points made through the linguistic considerations are based on the idea that a language phenomenon needs to be evaluated with regards to its interactions with other phenomena in the same language. There could be opaque processes which require referring to the derivational history of a construction such as nominalization in embeddings, argument dropping (subject, object, indirect object), compound making strategies, or grammatical functions of a clitic. Additionally, a language does not need to employ a structural property uniformly in its grammatical system. Classifiers in Turkish could be an example for this. Example sentences for the UD tagset could already exist in the provided guidelines, but they lack linguistic diagnostics which are crucial to differentiate between the closely related constructions and the mostly opaque processes in a given language. We hope explicitly stating the diagnostics used for an annotation scheme becomes a practice so that the unification process of the treebanks does not follow from standalone examples but rather from testable predictions.

4.4. Experiments

We performed parsing experiments on the BOUN Treebank as well as on its different text types, which will serve as a baseline for future studies. In addition to the brand-new BOUN Treebank, we experimented with our re-annotated versions of the IMST [80] and PUD [81] treebanks, in order to observe the effect of using additional training and test data.

Most prior studies [2, 6, 60, 75, 105, 106] on Turkish dependency parsing evaluate the treebanks they use (mostly versions of the IMST Treebank) using MaltParser [71]. However, the definition of a well-formed dependency tree for MaltParser is different than the conventions of UD such that the root node may have more than one child in the output of the MaltParser. UD defines a dependency tree with exactly one root node, and it is not possible to have MaltParser produce dependency trees that follow the UD convention. For this reason, we use a state-of-the-art graph-based neural parser [82] in the experiments. This parser uses unidirectional LSTM modules to

generate character-based word embeddings and bidirectional LSTM modules to create possible head-dependency relations. It uses ReLu layers and biaffine classifiers to score these relations. For more information, see [82].

As stated in Section 4.3, the BOUN Treebank consists of 9,761 sentences from five different text types. These text types almost equally contribute to the total number of sentences. For the parsing experiments, we randomly assigned each section to the training, development, and test sets with 80%, 10%, and 10% percentages, respectively. Table 4.8 shows the number of sentences in each set of the BOUN Treebank.

Table 4.8. Division of the BOUN Treebank and its different sections among training, development, and test sets for the experiments.

Treebank	Training set	Development set	Test set	Total
Essays	1,561	196	196	1,953
Broadsheet National Newspapers	1,518	190	190	1,898
Instructional Texts	1,580	198	198	1,976
Popular Culture Articles	1,568	197	197	1,962
Biographical Texts	1,576	198	198	1,972
BOUN	7,803	979	979	9,761

In order to observe the parsing performance for different types of text, we first evaluated the dependency parser for each section separately. Then, we measured the performance of the parser on parsing the entire BOUN Treebank. As a final set of experiments, we trained the parser on the training sets of the BOUN Treebank and the re-annotated version of the IMST Treebank separately and together, and tested them on five different settings. With that set of experiments, we aim to measure the difference in performance between the BOUN Treebank and the IMST Treebank and to observe the effect of increasing the training data size on performance for Turkish dependency parsing.

In our experiments, we did not perform pre-processing actions such as removing sentences that include non-projective dependencies from the training or test sets. All sentences in the treebanks were included in the experiments. As for the pretrained word vectors used by the dependency parser, we used the Turkish word vectors supplied by the CoNLL-17 organization [107]. For the evaluation of the dependency parser, we used the unlabeled attachment score (UAS) and labeled attachment score (LAS) metrics. In the experiments, we used gold POS tags instead of automatic predictions of them.

4.5. Results

4.5.1. Parsing Results on the BOUN Treebank

Table 4.9 shows the parsing results of the test sets for each section in the BOUN Treebank and the BOUN Treebank as a whole in terms of UAS and LAS. In these experiments, the parser has been trained by using the entire training set of the BOUN Treebank. We observed that the highest and lowest LAS were obtained on *Broadsheet National Newspapers* and *Essays* sections of the BOUN Treebank, respectively. The parser achieved more or less similar performance on the remaining three sections.

Table 4.9. UAS and LAS scores of the parser on the BOUN Treebank.

Treebank	UAS F1-score	LAS F1-score
Essays	68.73	59.18
Broadsheet National Newspapers	81.59	76.04
Instructional Texts	79.22	72.65
Popular Culture Articles	77.69	71.13
Biographical Texts	80.28	73.68
BOUN Treebank	77.36	70.37

To understand the possible reasons behind the performance differences between the parsing scores of the five sections of the BOUN Treebank, we compared the sections

with respect to the average token count and the average dependency arc length in a sentence. Figure 4.1 demonstrates these statistics for the five sections of the BOUN Treebank.⁴ We observed that both the average token count and the average dependency arc length metrics are the highest in the *Broadsheet National Newspapers* section. The second highest in both metrics is the *Essays* section. The averages for the *Instructional Texts*, *Popular Culture Articles*, and *Biographical Texts* sections are close to each other.

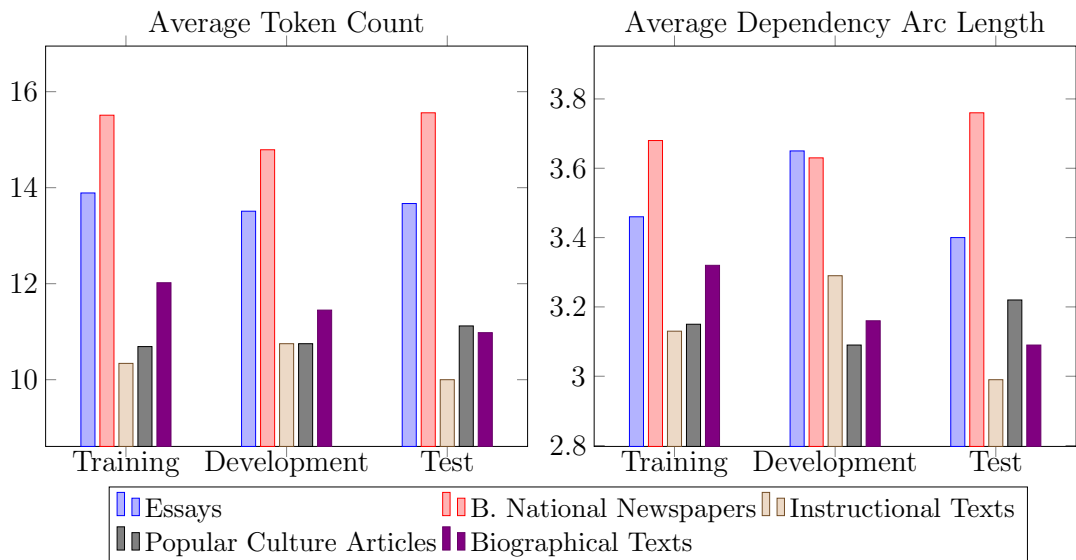


Figure 4.1. The average token count and the average dependency arc length in a sentence for the five sections of the BOUN Treebank.

Note that, *the average token count* metric, which shows the length of a sentence, and *the average dependency arch length* metric, which depicts the distance between the nodes of the dependency relations in a sentence, can sometimes correlate, although not all long sentences include long range dependencies. We anticipate that the higher these two metrics are in a sentence, the harder the task of constructing the dependency tree of that sentence will be. In Figure 4.1, we observe that all of the sections except the *Broadsheet National Newspapers* conform with this hypothesis. However, the *Broadsheet National Newspapers*, which has the highest numbers of these metrics holds the best parsing performance in terms of the UAS and LAS metrics. We believe that these

⁴The images within the scope of this thesis whose copyrights were transferred to the publishing companies were used in accordance with the publishing policies of publishers about the reuse of the text and graphics produced by the author.

high scores in this section are due to the lack of interpersonal differences in writing in journalese and the editorial process behind the journals and magazines.

4.5.2. Parsing Results on Combinations of Treebanks

In Table 4.10, we present the success rates of the parser trained and tested on different combinations of the three Turkish treebanks: the BOUN Treebank and the re-annotated versions of the IMST and Turkish PUD treebanks. We chose to include only these two treebanks that we re-annotated because we wanted to measure the effect of our unification efforts for Turkish treebanking on the parsing accuracy.

Table 4.10. The performance of the parser on five different test sets according to UAS and LAS metrics. On each test set, performance of the parser in the following settings is measured: when trained using only the IMST Treebank, when trained using only the BOUN Treebank, and when trained using these two treebanks together.

Training set	Training size	Test set	Test size	UAS F1-score	LAS F1-score
IMST	3,685	BOUN	979	69.38	58.65
BOUN	7,803	BOUN	979	77.36	70.37
BOUN+IMST	11,488	BOUN	979	77.57	70.50
IMST	3,685	IMST	975	75.49	65.53
BOUN	7,803	IMST	975	73.63	62.92
BOUN+IMST	11,488	IMST	975	76.86	66.79
IMST	3,685	PUD	1,000	65.28	49.50
BOUN	7,803	PUD	1,000	72.33	59.57
BOUN+IMST	11,488	PUD	1,000	72.76	60.39
IMST	3,685	BOUN+IMST	1,954	71.89	61.62
BOUN	7,803	BOUN+IMST	1,954	75.67	66.99
BOUN+IMST	11,488	BOUN+IMST	1,954	77.25	68.82
IMST	3,685	BOUN+IMST+PUD	2,954	69.03	56.37
BOUN	7,803	BOUN+IMST+PUD	2,954	74.22	63.78
BOUN+IMST	11,488	BOUN+IMST+PUD	2,954	75.30	65.17

The parser is trained separately on the training sets of the IMST and BOUN treebanks, and then, by combining these two training sets (denoted as BOUN+IMST in the first column of Table 4.10). Originally created for evaluation purposes [73], the PUD Treebank is not used in the training phase of these experiments due to its smaller size compared to the other two treebanks; instead, it is used as an additional test set in the evaluations.

Five different test sets are provided in the third column of Table 4.10: the test set of the BOUN Treebank (BOUN), the test set of the IMST Treebank (IMST), the Turkish PUD Treebank (PUD), the combined test sets of the BOUN and IMST treebanks (BOUN+IMST), and the combined test sets of the BOUN and IMST treebanks and the PUD Treebank (BOUN+IMST+PUD).

Each of the trained models is tested on these five test sets. We observe the following:

- The parser model trained on the BOUN Treebank outperforms the one trained on IMST by at least 10% in LAS on the first and third test sets (and $\sim 5\%$ on the fourth and fifth sets). Not surprisingly, the parser trained on IMST performs better on its own test set (the second test set) than the parser model trained on the BOUN Treebank. However, the performance difference here is smaller than the one when these two models are tested on the BOUN Treebank’s test set. To make a comparison, the parser trained on BOUN outperforms the parser trained on IMST by $\sim 8\%$ in UAS and by more than 10% in LAS when tested on the BOUN test set. On the other hand, for the case of the IMST test set, the parser trained on IMST outperforms the parser trained on BOUN by only $\sim 2\%$ in UAS and LAS. Having less amount of training data and a more inconsistent annotation history might be the cause of the inferior performance of the IMST Treebank when compared to the BOUN Treebank.
- Joining the training sets of the BOUN and IMST treebanks improves parsing performance in terms of the attachment scores. The increase in the training size

resulted in better parsing scores, contributing to the discussion on the correlation between the size of the corpus and the success rates in parsing experiments [54, 108].

- The worst results by all the models were obtained on the PUD Treebank used as a test set. The different nature of the PUD Treebank compared to the other Turkish treebanks may have an effect on this performance drop. This treebank includes sentences translated from different languages by professional translators and hence, the sentences have different structures than the sentences of the other two treebanks. This difference in structures is a result of the different environments in which these texts were brewed, namely a living corpus (BOUN and IMST) and well-edited translations (PUD).

4.5.3. Changes in Dependency Label Distribution

In order to investigate the differences in the percentages of certain dependency relations between the treebanks used in the experiments, we present the distribution of the dependency relation types across the previous⁵ as well as the re-annotated versions of the IMST and PUD treebanks, and the BOUN Treebank in Table 4.11.

When comparing the BOUN Treebank and the re-annotated version of the IMST Treebank, we observed that the percentages of the **case**, **compound**, and **nmod** types were lower by more than 1% in the BOUN Treebank. The percentage of the **root** type was also lower in the BOUN Treebank by almost 2%, which indicates that the average token count in sentences is higher in this treebank with respect to the re-annotated version of the IMST Treebank. However, the percentage of the **nmod:poss** type was higher by more than 2% and the **obl** type was higher by more than 3% in the BOUN Treebank. We believe that these differences are due to the text types we utilized. Unlike IMST, the BOUN Treebank includes essay and autobiography text types. These types make frequent use of postpositional phrases such as *bana göre* (*in my opinion*) or *1920'ye kadar* (*until 1920*), which are encoded with **case** dependency

⁵The re-annotation process was performed on the UD 2.3 versions of these treebanks.

relations. Additionally, the language is less formal compared to the non-fiction and news text types, which are the main registers that the IMST Treebank incorporates as indicated in the UD Project. This formality difference explains the lower usage of the `compound` relation type.

When comparing the BOUN Treebank with the re-annotated version of the Turkish PUD Treebank, we observed that the highest percentage difference was for the `obl` type which is higher in the BOUN Treebank by more than 7%. This difference is again a result of using different text types. The Turkish PUD Treebank consists of Wikipedia articles in which the adjuncts are expected to be used less than the text types we utilized. The other relation types whose percentages are higher in BOUN by more than 1% were the `root` type which indicates that the average token count is lower in the BOUN Treebank, and the `conj` type indicating that the BOUN Treebank has more conjunct relations which sometimes increased the complexity of a sentence in terms of dependency parsing.

In the comparison of the previous and re-annotated versions of the IMST Treebank with respect to the distribution of dependency relation types, we see that the percentages of the `advmod`, `cc`, `ccomp`, and `nsubj` types increased by approximately 1% in the re-annotated version. In contrast, the percentage of `nmod` is reduced by more than 3% in the re-annotated version. The reason behind this decrease lies in the fact that in the previous version of the treebank, nominalized verbs which behave like converbs [32] are considered nominal modifiers. However, these nominalized verbs actually construct embedded clauses and therefore are treated as clausal modifiers in the re-annotated treebank. In addition, the `obl` percentage decreased by more than 1% in the re-annotated version.

The `vocative` type no longer exists in the re-annotated version and the newly introduced types that are absent in the previous version are the `advcl`, `advcl:cond`, `aux`, `cc:preconj`, `clf`, `dislocated`, `goeswith`, `iobj`, `orphan`, and `xcomp` relation labels.

Table 4.11. Dependency label distribution of the BOUN Treebank together with previous and re-annotated versions of IMST and PUD treebanks. Black and gray numbers show counts and percentages, respectively.

Relation type	IMST (previous)	IMST (re-annotated)	PUD (previous)	PUD (re-annotated)	BOUN
acl	1,455 (2.5%)	1,538 (2.65%)	-	515 (3%)	3,494 (2.85%)
acl:relcl	-	-	514 (3.04%)	-	-
advcl	-	926 (1.59%)	405 (2.4%)	435 (2.6%)	2,595 (2.12%)
advcl:cond	-	110 (0.19%)	-	13 (0.07%)	269 (0.22%)
advmod	1,872 (3.2%)	2,422 (4.17%)	1,716 (10.16%)	1,624 (9.6%)	5,278 (4.31%)
advmod:emph	973 (1.67%)	976 (1.68%)	145 (0.86%)	143 (0.8%)	1,724 (1.41%)
amod	3,451 (5.94%)	3,337 (5.74%)	1,224 (7.25%)	1,318 (7.8%)	7,869 (6.43%)
appos	51 (0.09%)	136 (0.23%)	36 (0.21%)	166 (1%)	506 (0.41%)
aux	-	1 (0.002%)	21 (0.12%)	4 (0.02%)	39 (0.03%)
aux:q	209 (0.36%)	211 (0.36%)	-	1 (0.01%)	269 (0.22%)
case	2,183 (3.76%)	2,242 (3.86%)	694 (4.1%)	697 (4.1%)	3,290 (2.69%)
cc	870 (1.5%)	879 (3.1%)	519 (3.1%)	520 (3.1%)	2,800 (2.29%)
cc:preconj	-	3 (0.005%)	8 (0.05%)	8 (0.05%)	134 (0.11%)
ccomp	36 (0.06%)	626 (1.08%)	30 (0.18%)	171 (1%)	1,512 (1.24%)
clf	-	8 (0.01%)	10 (0.06%)	10 (0.06%)	122 (0.1%)
compound	2219 (3.82%)	1,977 (3.40%)	2012 (11.91%)	314 (1.9%)	2,381 (1.95%)
compound:lvc	512 (0.88%)	522 (0.90%)	-	186 (1.1%)	1,218 (1.0%)
compound:redup	199 (0.34%)	219 (0.37%)	-	9 (0.05%)	457 (0.37%)
conj	3,718 (6.40%)	3,529 (6.07%)	640 (3.79%)	696 (4.1%)	7,250 (5.92%)
cop	813 (1.40%)	851 (1.46%)	517 (3.06%)	496 (2.9%)	1,289 (1.05%)
csubj	7 (0.01%)	82 (0.14%)	115 (0.68%)	93 (0.5%)	546 (0.45%)
dep	1 (0.002%)	1 (0.002%)	3 (0.02%)	3 (0.02%)	9 (0.01%)
det	2,040 (3.51%)	1,975 (3.39%)	671 (3.97%)	680 (4%)	4,938 (4.03%)
det:predet	-	-	10 (0.06%)	8 (0.05%)	-
discourse	154 (0.27%)	150 (0.26%)	5 (0.03%)	5 (0.03%)	381 (0.31%)
dislocated	-	20 (0.03%)	2 (0.01%)	5 (0.03%)	28 (0.02%)
fixed	40 (0.07%)	25 (0.04%)	204 (1.21%)	1 (0.01%)	12 (0.01%)
flat	910 (1.57%)	902 (1.55%)	4 (0.02%)	409 (2.4%)	2,039 (1.67%)
flat:name	-	-	247 (1.46%)	-	-
goeswith	-	3 (0.005%)	1 (0.01%)	1 (0.01%)	4 (0.002%)
iobj	-	354 (0.61%)	90 (0.53%)	138 (0.8%)	164 (0.13%)
list	-	-	-	-	40 (0.03%)
mark	76 (0.13%)	86 (0.15%)	6 (0.03%)	5 (0.03%)	117 (0.10%)
nmod	3,780 (6.51%)	1,870 (3.22%)	161 (0.95%)	174 (1%)	1,371 (1.12%)
nmod:arg	-	-	110 (0.65%)	-	-
nmod:poss	3,534 (6.08%)	3,598 (6.19%)	722 (4.27)	1,881 (11%)	10,393 (8.49%)
nsubj	3,747 (6.45%)	4,430 (7.63%)	1,023 (6.05%)	1,238 (7.3%)	8,499 (6.94%)
nummod	621 (1.07%)	567 (0.98%)	207 (1.22%)	263 (1.6%)	1,568 (1.28%)
obj	4,307 (7.41%)	3,743 (6.44%)	816 (4.83%)	945 (5.6%)	7,381 (6.03%)
obl	4,444 (7.65%)	3,824 (6.58%)	148 (0.88%)	412 (2.4%)	12,015 (9.82%)
obl:tmod	-	-	232 (1.37%)	-	-
orphan	-	12 (0.02%)	12 (0.07%)	8 (0.05%)	84 (0.07%)
parataxis	11 (0.02%)	11 (0.02%)	74 (0.44%)	15 (0.09%)	209 (0.17%)
punct	10,228 (17.61%)	10,257 (17.65%)	2,150 (12.72%)	2,148 (12.7%)	20,116 (16.44%)
root	5,635 (9.69%)	5,635 (9.69%)	1,000 (5.91%)	1,000 (5.91%)	9,761 (7.97%)
vocative	1 (0.002%)	-	1 (0.001%)	-	88 (0.07%)
xcomp	-	39 (0.07%)	381 (2.26%)	125 (0.7%)	125 (0.1%)
Total	58,097	58,098	16,886	16,886	122,384

When we analyze the differences between the previous and re-annotated versions of the PUD Treebank, we observe that the biggest difference is in the `compound` relation with a 10% reduction. On the other hand, the biggest increase in the percentage of a relation is in the `nmod:poss` relation with a more than 6% increase in the re-annotated version. This is because in the previous annotation of the PUD Treebank, some constructions that involve genitive-possessive suffixes are marked with the `compound` dependency label. Such relations have been corrected as `nmod:poss`. Other noteworthy differences are in the `fixed` and `xcomp` relations with a more than 1% decrease and in the `flat`, `nsubj`, and `obl` relations with a more than 1% increase in the re-annotated treebank.

4.6. Conclusion

In this chapter, we presented the largest and the most comprehensive Turkish treebank with 9,761 sentences: the BOUN Treebank. In the treebank, we encoded the surface forms of the sentences, the universal part of speech tags, lemmas, and morphological features for each segment, as well as the syntactic relations between these segments. We explained our annotation methodology in detail. We also gave an overview of other Turkish treebanks. Moreover, we explained our linguistic decisions and annotation scheme that are based on the UD framework. We provided examples for the challenging issues that are present in the BOUN Treebank as well as other treebanks that we re-annotated. Our treebank with a history of the changes we applied and our annotation guidelines are provided online.

Lastly, we evaluated our new treebank on the task of dependency parsing. We reported UAS and LAS F1-scores with regards to specific text types and treebanks. We also showcased the results of the experiments where our new treebank was used with the re-annotated versions of the IMST and PUD treebanks. All the tools and materials that are presented in this chapter are available in [13].

5. A MORPHOLOGY-BASED REPRESENTATION MODEL FOR DEEP DEPENDENCY PARSING

5.1. Introduction

In this chapter, we present our initial study on dependency parsing. We introduce morphologically enhanced character-based word embeddings to improve the parsing performance especially for agglutinative languages. We apply our approach to a transition-based dependency parser by Ballesteros *et al.* [109] that uses stack-LSTM structures to predict the parser state. This parser uses character-level word representation, which has been shown to perform better for languages with rich morphology [82, 109]. We conducted experiments on UD version 2.2 datasets [110] which comprise 82 test sets from 57 languages. We observe that including morphological information to a character-based word embedding model yields a better learning of relationships between words and increases the parsing performance for most of the agglutinative languages with rich morphology.

This chapter is based on the work published as [111] which describes our submission to the CoNLL 2018 shared task [112] on parsing of Universal Dependencies (UD) [4].

5.2. The Parsing Model

We use the LSTM-based parser by Ballesteros *et al.* [109]. This parser is an improved version of a state-of-the-art transition-based dependency parser proposed by Dyer *et al.* [40] and uses stack-LSTM structures with push and pop operations to learn representations of the parser state. Instead of lookup-based word representations, BiLSTM modules are used to create character-based encodings of words. With this character-based modeling, the authors obtain improvements on the dependency parsing of many morphologically rich languages.

5.2.1. Character Embeddings of Words

The character-based word embedding model using BiLSTMs in [109] is depicted in Figure 5.1. The authors compute character-based vector representations of words using BiLSTMs. Their embedding system reads each word character by character from the beginning to the end and computes an embedding vector of the character sequence, which is denoted as \vec{w} in Figure 5.1. The system also reads the word character by character from the end to the beginning and the produced embedding is denoted as \overleftarrow{w} . These two embedding vectors and the learned representation of the POS tag t of the word are concatenated to produce the vector representation of the word. A linear mapping of POS tag words to integers is used to create a representation of the POS tags as in [109].

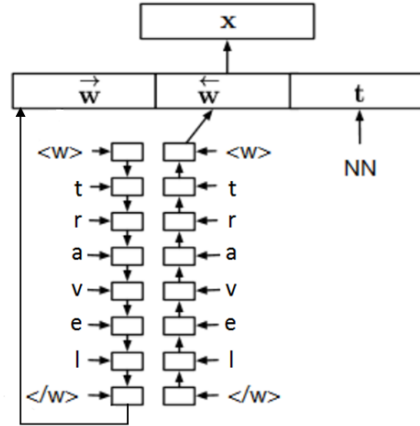


Figure 5.1. Vector representation of the word *travel* with the character-based embedding model in [109].

5.2.2. Morphology-based Character Embeddings

To improve the parsing performance of the LSTM-based parser [109] through the character-based word embeddings mentioned in Section 5.2.1, we include the morphological information of words to this embedding model. In agglutinative languages like Turkish, a stem usually takes different suffixes and by this way, different meanings

are created using a single root-word. Words that share the same suffixes tend to have similar roles in a sentence. For instance, gerunds in Turkish are a kind of derivational suffixes. Verbs that take the same gerund as a suffix have usually the same role in sentences. Table 5.1 shows some statistics of verbs with gerunds in the development dataset of the Turkish IMST Treebank for demonstration purposes. The first column shows some example suffixes that attach to verbs and turn them to adverbs. The second column shows the number of verbs with the corresponding suffix in the development set. The third column shows the statistics of the dependency labels of these verbs. As it can be seen from the table, these suffixes help determining the role of the word they attach to. Therefore, representing each word using its corresponding lemma and suffixes separately and utilizing the morphological information of words can improve the parsing performance in agglutinative languages.

Table 5.1. Number of occurrences of some example suffixes and the corresponding dependency labels of verbs with these suffixes in the development data of the IMST Treebank.

Suffix	Number of Occurrences	Dependency Label				
-Ip	41	23 nmod	8 compound	5 conj	4 obj	1 root
-ArAk	32	26 nmod	3 conj	2 compound	1 root	
-ken	20	18 nmod	1 conj	1 acl		
-IncA	8	7 nmod	1 compound			
-mAdAn	7	4 nmod	2 compound	1 obj		
-DIkçA	3	3 nmod				

5.2.2.1. Lemma-Suffix Model. For agglutinative languages where the stem of a word is not modified in different word forms, we created a model that uses lemma and suffix information of words in character-based embeddings. In this model, each word is separated to its lemma and suffixes. Then, the embedding system first reads the lemma of the word character by character from the beginning to the end and computes

an embedding vector of the character sequence of the lemma which is denoted as \vec{r} . Secondly, the system reads the lemma character by character from the end to the beginning and the produced embedding is denoted as \overleftarrow{r} . A similar process is performed for the suffixes of the word and the produced vectors are denoted as \vec{s} and \overleftarrow{s} . These four embedding vectors and the vector representation of the POS tag t of the word are then concatenated to produce the vector representation of the word. POS tag representations are created by linearly mapping the POS tags to integers as in [109]. Vector representation of an example word using this model is depicted in Figure 5.2.

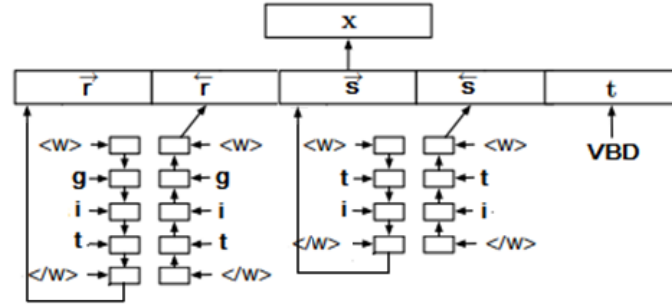


Figure 5.2. Character-based word embedding of a Turkish word *gitti* (“he/she/it went” in English) using Lemma-Suffix Embedding Model.

5.2.2.2. Morphological Features Model. The Lemma-Suffix Model is suitable only for agglutinative languages which make use of suffixes to create different word forms. For languages that do not have this type of grammar, we created another model where the specific morphological features of each word are embedded to the dense representations of the words. The reason behind this choice is that some morphological features have a direct impact in identifying the dependency labels of words. For instance, if a word has a *case* feature and its value is *accusative*, then it is usually an object of the sentence. By extracting and utilizing such morphological features, we can improve the parsing accuracy for languages that suit this model.

In this model, the embedding of a word is created character by character as in Section 5.2.1. Then, the embedding vector of each of its selected morphological features

are created by reading the feature value character by character from the beginning to the end. Finally, these embedding vectors and the vector representation of the POS tag of the word are concatenated to produce the vector representation of the word.

The vector representation of an example word using its morphological features is shown in Figure 5.3.

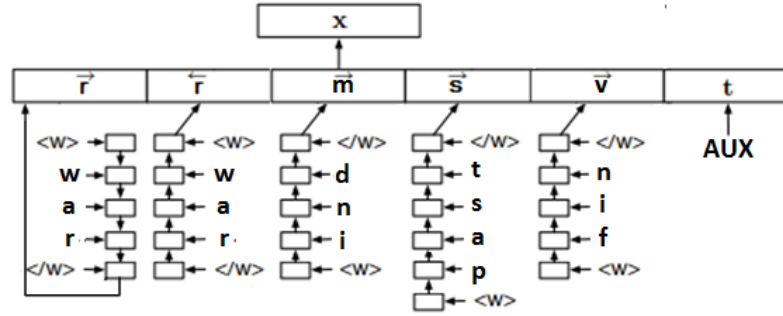


Figure 5.3. Character-based word embedding of a German word *war* (“was” in English) when the Morphological Features Embedding Model is used. The morphological features of the word *war* are: Mood=Ind, Number=Sing, Person=3, Tense=Past, and VerbForm=Fin. Since the Morphological Features Embedding Model utilizes only the Case, Mood, Tense, and VerbForm features for German, only the values of these features are embedded. Note that the Case feature is represented with an empty string in the word vector of *war* as there is no Case feature in its morphological features.

5.3. Experiments

To evaluate our models, we performed a set of experiments on the test data of UD version 2.2 treebanks. The purpose of these experiments is to investigate the effect of our embedding models on parsing performance. Instead of parsing raw texts, we used the gold-standard CoNNL-U files [113] in the experiments. The following sections give details of the experimental setup.

5.3.1. Extracting Lemmas and Suffixes

The Lemma-Suffix Embedding Model needs to have lemma and suffix information for each input word. For this purpose, we have two approaches. The classic and more accurate approach is to use a morphological analyzer and disambiguator tool to find lemmas and suffixes of words. The other more simpler approach is using the lemma column of the CoNNL-U files of the treebanks and removing the corresponding lemma from each word to find the suffix information. These two methods are shown on an example sentence in Table 5.2. We compared these two approaches on the development set of the IMST Treebank. For this purpose, we utilized the Turkish morphological parser and disambiguator by Sak *et al.* [85]. We observed that finding suffixes by removing lemmas from the words gives the same parsing performance as using a morphological analyzer tool to find the lemma and suffixes of a word. So, we opted not to use a morphological analyzer and disambiguator for the languages with the lemma-suffix embedding model due to additional costs of these tools.

Table 5.2. Lemma and suffix separation example without using morphological analyzer and disambiguator and with using morphological analyzer and disambiguator on the Turkish sentence “*Her şeyden önce sanatçıydı.*” (Sentence translation: “*She was an artist before anything else.*”)

	No morp.parser		With morp.parser	
Word	Lemma	Suffix	Lemma	Suffix
Her	her	-	her	-
şeyden	şey	den	şey	DAn
önce	önce	-	önce	-
sanatçıydı	sanat	çıydı	sanat	CHY DH

5.3.2. Embedding Model Selection for Different Languages

For each language, we decided which embedding model to apply according to the characteristics of that language.

Table 5.3. List of morphological features used for the languages with the Morphological Features Embedding Model.

Language	Morphological Features			
Afrikaans	Aspect	Case	Tense	VerbForm
Ancient Greek	Aspect	Case	Tense	VerbForm
Arabic	Aspect	Case	Mood	VerbForm
Armenian	Aspect	Case	Tense	VerbForm
Basque	Aspect	Case	Tense	VerbForm
Bulgarian	Aspect	Case	Tense	VerbForm
Catalan	AdpType	Mood	Tense	VerbForm
Croatian	Case	Mood	Tense	VerbForm
Czech	Aspect	Case	Tense	VerbForm
Dutch	Degree	Case	Tense	VerbForm
English	Case	Mood	Tense	VerbForm
Estonian	Case	Mood	Tense	VerbForm
French	Mood	Tense	VerbForm	
Finnish	Case	Mood	Tense	VerbForm
Galician	Case	Mood	Tense	VerbForm
German	Case	Mood	Tense	VerbForm
Gothic	Case	Mood	Tense	VerbForm
Greek	Aspect	Case	Tense	VerbForm
Hebrew	HebBinyan	HebSource	Tense	VerbForm
Hindi	Aspect	Case	Tense	VerbForm
Indonesian	PronType	Degree		
Irish	Case	Mood	Tense	VerbForm
Italian	PronType	Mood	Tense	VerbForm
Kurmanji	Case	Mood	Tense	VerbForm
Latin	Case	Mood	Tense	VerbForm
Latvian	Aspect	Case	Tense	VerbForm
North Sami	Case	Mood	Tense	VerbForm
Norwegian	Case	Mood	Tense	VerbForm
Old Church Slavonic	Case	Mood	Tense	VerbForm
Old French	Tense	VerbForm		
Polish	Aspect	Case	Tense	VerbForm
Portuguese	PronType	Mood	Tense	VerbForm
Romanian	Case	Mood	Tense	VerbForm
Russian	Aspect	Case	Tense	VerbForm
Serbian	PronType	Mood	Tense	VerbForm
Slovak	Aspect	Case	Tense	VerbForm
Slovenian	Aspect	Case	Tense	VerbForm
Spanish	Case	Mood	Tense	VerbForm
Swedish	Case	Mood	Tense	VerbForm
Ukrainian	Aspect	Case	Tense	VerbForm
Upper Sorbian	Case	Mood	Tense	VerbForm
Mixed Language	Case	Mood	Tense	VerbForm

We applied the Lemma-Suffix Model in 5.2.2.1 to Buryat, Hungarian, Kazakh, Turkish, and Uyghur languages because these languages have agglutinative morphology, take suffixes, and the stem of a word usually does not change in different word forms. We also applied this model to Danish to observe the effect in parsing performance of a language with little inflectional morphology.

For the languages that do not follow this scheme, we applied the Morphological Features Embedding Model in 5.2.2.2. Table 5.3 shows the morphological features selected for these languages. We selected four morphological features from the input CoNNL-U files for most of the languages. For French, Indonesian, and Old French, we used less than four features because there are less than four common morphological features in the CoNNL-U files of these languages.

For Persian, Japanese, Korean, Vietnamese, and Chinese, we used the baseline embedding model due to the lack of representative morphological features in their corresponding CoNNL-U files.

Languages without Training Data. We trained a mixed language parser model with the Morphological Features Model for languages with no training data. To train the model, we used a mixed language data which includes the first 200 sentences of each treebank. This model is applied to the Buryat KEB, Czech PUD, English PUD, Faroese OFT, Japanese Modern, Naija NSC, Swedish PUD, and Thai PUD treebanks.

5.3.3. Training Specifications

Our model mostly uses the same hyper-parameter configuration with the original settings in [109] with a few exceptions. We used stochastic gradient descent trainer with a learning rate of 0.13. We replaced the base embedding model with our embedding models. In the Lemma-Suffix Model, the forward and backward word vectors of the lemma of a word both have 50 dimensions. The forward and backward word vectors of the suffix of a word also have 50 dimensions each. In the Morphological Features Model,

each of the forward and backward vectors of a word have 50 dimensions. Each of the four morphological feature vectors have 25 dimensions. If a morphological feature is absent in a word, an embedding vector of an empty string is created for that feature. So, we increased the dimension of the character-based representations to 200 in total.

The original parser is not compatible with UD parsing. We adapted it to be able to take input and produce output in CoNNL-U format. The source code of our modified version of the LSTM-based parser in [109] can be found at [114].

5.4. Results

Table 5.4 shows a comparison of our models with the baseline model [109] in terms of LAS, MLAS, and BLEX scores on these datasets.

From the comparative results shown in Table 5.4, we observe that out of 62 datasets, our models outperform the baseline on 37 of them. We notice that the Morphological Features Model outperforms the baseline model on the languages that have rich inflectional and derivational processes mostly by adding suffixes to words. This is the case for the Bulgarian, Croatian, Czech, Basque, Gothic, Latin, Polish, Russian, Slovak, Slovene, North Sami, and Ukrainian languages.

The Morphological Features Model is not suitable for the grammatical structure of Arabic, which has derivational morphology. It also fails to outperform baseline in Romanic languages like French, Spanish, Catalan, Galician, and Portuguese. The possible reason for this might be the analytic structure of these languages where every morpheme is an independent word. English, Hebrew, Hindi and Urdu are also categorized as mostly analytic languages which do not use inflections and have a low morpheme-per-word ratio [115]. Dutch, Norwegian, and Swedish have a simplified inflectional grammar and cannot be represented well using our model. Besides, our model is not the best choice for languages that have high ratio of morphophonological modifications to the root word like Old Church Slavonic.

Table 5.4. Comparison of our models with the baseline. MF and LS stand for the Morphological Features and Lemma-Suffix models, respectively. Green(*Red*) rows show scores of treebanks on which our models outperform(*fall behind*) the baseline.

Treebank	Emb. model	LAS		MLAS		BLEX	
		Baseline	Our model	Baseline	Our model	Baseline	Our model
<i>af-afribooms</i>	MF	82.16	82.80	73.17	74.35	75.48	76.37
<i>ar-padt</i>	MF	78.80	78.60	73.59	73.33	74.66	74.39
<i>bg-btb</i>	MF	86.52	87.41	80.88	82.00	81.33	82.38
<i>ca-ancora</i>	MF	87.21	87.03	80.56	80.23	81.07	80.79
<i>cs-cac</i>	MF	87.37	87.85	84.12	84.94	84.78	85.49
<i>cs-fictree</i>	MF	83.49	86.03	77.80	81.64	78.58	82.34
<i>cs-pdt</i>	MF	86.66	88.47	83.39	85.89	83.91	86.38
<i>cu-proiel</i>	MF	75.73	75.59	69.85	69.62	72.09	71.97
<i>da-ddt</i>	LS	77.34	78.04	71.45	71.48	72.97	73.33
<i>de-gsd</i>	MF	77.45	77.79	69.79	70.26	72.40	73.24
<i>el-gdt</i>	MF	83.22	83.98	74.83	76.69	75.57	77.53
<i>en-ewt</i>	MF	83.88	83.58	79.44	78.95	79.96	79.51
<i>en-gum</i>	MF	80.34	81.46	73.30	74.34	73.87	75.03
<i>en-lines</i>	MF	75.89	73.83	71.06	67.75	72.53	69.28
<i>es-ancora</i>	MF	86.71	85.55	80.65	78.97	81.17	79.61
<i>et-edt</i>	MF	80.25	81.49	76.59	78.28	77.40	78.98
<i>eu-bdt</i>	MF	74.07	74.65	69.97	71.21	71.69	72.74
<i>fi-ftb</i>	MF	82.76	83.88	77.86	79.02	78.54	79.89
<i>fi-tdt</i>	MF	80.39	80.46	76.33	76.60	76.96	77.31
<i>fr-gsd</i>	MF	84.69	83.77	78.56	77.59	79.13	78.39
<i>fr-sequoia</i>	MF	83.16	82.49	76.75	75.96	77.38	76.40
<i>fr-spoken</i>	MF	67.99	68.70	57.82	58.19	58.68	59.04
<i>fro-srcmf</i>	MF	83.01	82.54	76.90	76.44	77.78	77.43
<i>ga-idt</i>	MF	61.02	63.23	45.21	47.98	48.68	51.90
<i>gl-ctg</i>	MF	81.56	80.70	70.76	69.41	75.38	74.25
<i>got-proiel</i>	MF	71.88	74.95	64.13	67.99	66.78	70.78
<i>grc-perseus</i>	MF	61.22	60.10	50.75	49.95	53.92	52.79
<i>grc-proiel</i>	MF	79.26	79.28	64.54	64.12	67.09	67.16
<i>he-htb</i>	MF	80.06	79.80	71.56	70.97	72.02	71.52
<i>hi-hdtb</i>	MF	92.11	91.51	87.64	86.72	88.38	87.46
<i>hr-set</i>	MF	80.12	81.36	74.75	76.45	76.22	77.90
<i>hu-szeged</i>	LS	64.00	68.33	56.44	62.55	59.75	66.11
<i>hy-armtdp</i>	MF	29.60	28.56	21.65	25.14	24.04	28.56
<i>it-isdt</i>	MF	88.91	89.23	82.77	83.34	83.20	83.79
<i>it-postwita</i>	MF	79.19	79.10	72.30	72.26	72.84	72.91
<i>kk-ktb</i>	LS	35.92	35.34	25.89	25.19	30.09	30.18
<i>la-itb</i>	MF	83.86	85.37	79.06	80.93	80.02	82.10
<i>la-perseus</i>	MF	47.48	51.82	41.00	46.56	44.56	51.79
<i>la-proiel</i>	MF	68.81	70.95	62.15	64.66	64.95	67.11
<i>lv-lvtb</i>	MF	73.48	75.43	66.19	68.67	67.37	69.66
<i>nl-alpino</i>	MF	80.60	79.11	72.53	70.60	73.25	71.44
<i>nl-lassysmall</i>	MF	81.15	79.19	74.84	72.37	75.52	73.26
<i>no-bokmaal</i>	MF	88.53	88.22	84.48	83.87	84.96	84.46
<i>no-nynorsk</i>	MF	86.64	85.42	82.00	80.39	82.94	81.21
<i>no-nynorskliia</i>	MF	66.27	64.76	58.40	56.92	60.12	58.55
<i>pl-lfg</i>	MF	92.02	92.68	88.93	89.80	89.16	90.01
<i>pl-sz</i>	MF	85.86	89.56	81.34	86.50	82.02	87.17
<i>pt-bosque</i>	MF	83.28	83.20	75.64	74.85	76.95	76.28
<i>ro-rrt</i>	MF	81.22	80.84	74.26	74.03	75.55	75.36
<i>ru-syntagrus</i>	MF	88.01	88.14	84.35	84.86	84.72	85.24
<i>ru-taiga</i>	MF	48.95	56.57	40.82	50.74	42.74	52.33
<i>sk-snk</i>	MF	78.49	82.66	73.94	79.61	74.58	80.46
<i>sl-ssj</i>	MF	86.23	88.82	81.84	85.33	82.23	85.80
<i>sl-sst</i>	MF	64.47	65.41	57.67	59.38	59.31	61.22
<i>sme-giella</i>	MF	66.21	71.55	58.73	66.87	61.22	69.03
<i>sr-set</i>	MF	80.71	80.36	75.36	74.84	76.74	76.38
<i>sv-lines</i>	MF	76.86	77.43	72.98	73.75	74.13	74.72
<i>sv-talbanken</i>	MF	83.03	82.39	78.36	77.68	79.27	78.58
<i>tr-imst</i>	LS	55.45	56.74	49.29	50.42	50.45	51.97
<i>ug-udt</i>	LS	58.02	56.97	47.47	45.52	50.18	47.86
<i>uk-iu</i>	MF	76.10	78.87	70.57	74.66	70.77	74.95
<i>ur-udtb</i>	MF	86.07	86.04	79.43	79.77	80.75	81.02

The Lemma-Suffix Embedding Model is applied to the Danish, Hungarian, Kazakh, Turkish, and Uyghur languages. The best performance is reached in the Hungarian language with more than 4% increase in LAS score. Our model also outperforms the baseline in Turkish. These languages are highly agglutinative languages where words may consist of several morphemes and the boundaries between morphemes are clearcut. In this type of languages, there is a one-to-one form-meaning correspondence and shape of a morpheme is invariant [115]. An example word-morpheme relationship in Hungarian and Turkish languages is shown in Table 5.5. As it can be seen from the table, this structure is very suitable to the Lemma-Suffix Embedding Model.

Table 5.5. Word-morpheme structure of the Hungarian word *ember* and the Turkish word *adam* (English translation of these words: *man*).

	Singular	Plural	Singular	Plural
	<i>Hungarian</i>	<i>Hungarian</i>	<i>Turkish</i>	<i>Turkish</i>
Nominative	<i>ember</i>	<i>ember-ek</i>	<i>adam</i>	<i>adam-lar</i>
Accusative	<i>ember-et</i>	<i>ember-ek-et</i>	<i>adam-ı</i>	<i>adam-lar-ı</i>
Dative	<i>ember-nek</i>	<i>ember-ek-nek</i>	<i>adam-a</i>	<i>adam-lar-a</i>
Locative	<i>ember-ben</i>	<i>ember-ek-ben</i>	<i>adam-da</i>	<i>adam-lar-da</i>

Yet, the Lemma-Suffix Model fails to reach better performance than the baseline on the Kazakh and Uyghur treebanks. A possible reason might be that our embedding model increases the system complexity unnecessarily for these low-resource languages. The Lemma-Suffix Model outperforms the baseline for Danish, although it can be considered as an analytic language with a simplified inflectional grammar.

Table 5.6 shows the parsing scores of the parser with Lemma-Suffix Embedding Model on the test data of the Turkish IMST Treebank. We compared the parsing performance when the parser does not use pretrained word embeddings additional to the character embeddings, when it uses pretrained embeddings from CoNLL-17 UD word embeddings, and when it uses pretrained embeddings from word vectors trained

on Wikipedia by Facebook [116]. From the results, we observe that the usage of pre-trained word vectors increases the parsing performance by great extent for Turkish. We also observe that Facebook word vectors outperform the CoNLL-17 UD word vectors [107], although the number of words in the Facebook vectors data set is much smaller than the number of words in the CoNLL-17 UD word vectors data set.

Table 5.6. Effect of using pretrained word embeddings on the parsing success of Turkish IMST test set.

Treebank	Number of words	Embedding dimension	LAS	MLAS	BLEX
<i>tr-imst</i> w/o pretrained embeddings	-	-	56.74	50.42	51.97
<i>tr-imst</i> with CoNLL-17 UD word embeddings	3,633,786	100	59.11	53.02	54.51
<i>tr-imst</i> with Facebook word embeddings	416,051	300	59.69	53.56	54.98

5.5. Conclusion

In this preliminary study, we introduced two morphology-based adaptations of the character-based word embedding model in [109] and experimented with these models on the UD version 2.2 treebanks. The experimental results suggest that our models utilizing morphological information of words increase the parsing performance in agglutinative languages.

6. A HYBRID DEEP DEPENDENCY PARSER ENHANCED WITH RULES AND MORPHOLOGY

6.1. Introduction

Current state-of-the-art dependency parsers usually rely solely on deep learning methods, where parsers try to learn the characteristics of the language from available training data [40, 117]. As expected, this approach works well when the training data size is big enough. However, these pure deep learning-based approaches cannot reach the desired success levels when the data size is insufficient [118]. It was observed that deep learning-based systems need large amounts of data to be able to reach high performance [119]. For languages with small data sets, there is a need for developing additional methods that meet the characteristic needs of these languages.

Following our initial study in Chapter 5 on the use of morphology in dependency parsing, we propose to take into account the language grammar and integrate the information extracted from the grammar to a deep learning-based dependency parser. We propose two approaches for the inclusion of the grammar to the neural parser model. Our first approach is to integrate linguistically-oriented rules to a deep learning-based parser for dependency parsing of languages especially with restricted amount of training data. The rules are created to deal with the problematic parts in the sentences that are hard to predict. In our second approach, we give morpheme information as an additional input source to the parsing system. We experimented with different methods for inclusion of the morpheme information. We applied the proposed methods to Turkish and the experimental results suggest that both approaches improve the parsing performance of a state-of-the-art dependency parser for Turkish.

The proposed methods were evaluated on both projective and non-projective sentences and currently hold the state-of-the-art performance in parsing the Turkish IMST Treebank. To the best of our knowledge, this is the first study that integrates

the parsing actions of a rule-based method and morphological elements into a deep learning-based dependency parsing system and may serve as a base for other low- or mid-resource languages.

The main contributions presented in this chapter are as follows:

- A novel rule-based enhancement method that can be integrated to any neural dependency parser.
- A morphology-based enhancement method with three different ways of including morphological information to the parser.
- A simple yet useful integration method that allows to combine the proposed enhancement methods with any neural dependency parser.
- State-of-the-art dependency parsing scores on the IMST Treebank.

This chapter is based on our work in [9] and supported by the Scientific and Technological Research Council of Turkey (TÜBİTAK) under grant number 117E971 and as BİDEB 2211 graduate scholarship.

6.2. Related Work

Purely rule-based approaches to NLP problems have been very popular in the past, from part of speech tagging [120] to aspect extraction in sentiment analysis [121]. Rule-based methods have also been applied to dependency parsing. There have been studies on rule-based parsing using grammar rules for Turkish [122] and for other languages [123–126].

Deep learning methods show promising performance in predicting the dependency parses of sentences [40, 41, 117]. In 2017, a state-of-the-art LSTM-based biaffine parser [82] achieved the best performance in 54 treebanks including the IMST Treebank at the CoNLL-17 shared task on multilingual parsing from raw text to Universal Dependencies [73]. This parser together with its enhanced versions [84, 127] presented

at the CoNLL-18 shared task on multilingual parsing from raw text to Universal Dependencies [112] show state-of-the-art performance on the dependency parsing of many languages. However, these parsers do not have language specific features that can boost the parsing performance, especially for morphologically rich and under-resourced languages like Turkish.

Morphologically rich languages (MRLs) pose problems when state-of-the-art NLP models developed for the most widely studied languages like English and French are applied directly to them [128]. There are studies that include rule-based knowledge to data-driven parsers in order to increase parsing accuracy. Zeman and Žabokrtský [129] experimented with different voting mechanisms to combine seven different dependency parsers including a rule-based parser. Another study applied a rule-based mechanism on the output of a dependency parser to create collapsed dependencies [130].

There have also been several approaches that use morphological information in the dependency parsing of the MRLs. Similar to Ambati *et al.* [131] and Goldberg and Elhadad [132], which utilize morphological features for the dependency parsing of Hindi and Hebrew respectively, Marton *et al.* [133] measured the effects of nine morphological features extracted from an Arabic morphological analysis and disambiguation toolkit [134] on the parsing performance of the MaltParser [71] for Arabic. These studies show that usage of some morphological features works well for the dependency parsing of MRLs. Vania *et al.* [135] compared the strength of character-level modelling of words with an oracle model which has explicit access to morphological analysis of words on dependency parsing and observed that combining words with their corresponding morpheme information using a BiLSTM structure in the word representation layer outperforms the character-based word representation models. Dehouck and Denis [136] proposed a multi-task learning framework that makes use of language philogenetic trees to represent the shared information among the languages. They used gold morphological features for dependency parsing by summing the created vectors of each morphological attribute given in the treebanks and add this vector to the representation of the word, similarly to [111].

However, to the best of our knowledge, there does not exist any prior research on a hybrid approach for dependency parsing, where parsing decisions of hand-crafted rules together with morphological information are integrated into a deep learning-based dependency parsing model. Our inclusion of morphology also differs from the previous works in terms of extracting the morphological information. Instead of using morphological features of a word, our models utilize its suffixes explicitly. While two of the proposed morphology-based methods include the suffixes of each word to the word representation model directly, the third one represents each word with the suffixes which can and cannot bind to the root of that word.

Turkish Dependency Parsing. In this study, we propose both rule-based and morphology-based enhancement methods for dependency parsing and integrate our methods to a state-of-the-art dependency parser [82]. We applied the proposed approaches to Turkish. Because, unlike frequently studied languages such as English, the resources for Turkish NLP in general are restricted, which makes it suitable for such enhancements. Until very recently the only data sets used for training and evaluation of the systems for Turkish were the IMST Treebank [60] which consists of 5,635 annotated sentences and the Turkish UD Parallel (PUD) Treebank [73] of 1,000 annotated sentences that is used for testing purposes.

Turkish is not a well-studied language in natural language processing, and dependency parsing is no exception to this. Following the initial work in [122], another study presents a word-based and two inflectional group-based input representation models for dependency parsing of Turkish [137] which use a version of backward beam search to parse the sentences. They used a subset of 3,398 sentences of the Turkish Dependency Treebank [57] with only projective (non-crossing) dependency relations to train and test the proposed parsers. Later, a data-driven dependency parser for Turkish was proposed, which relies completely on inductive learning from treebank data for the analysis of new sentences, and on deterministic parsing for disambiguation [105]. The authors use a variant of the transition-based parsing system MaltParser proposed in [71], a linear-time, deterministic, classifier-based parsing method with history-based

feature models and discriminative learning.

Eryiğit *et al.* [138] extracted different multi-word expression classes as a pre-processing step for a statistical dependency parser. Only the projective dependencies are considered. Hall *et al.* [6] and Durgar El-Kahlout *et al.* [106] are other notable studies that are based on optimized versions of the MaltParser system. Later, a graph-based approach was proposed in [139], where a discriminative linear model is trained and a lattice dependency parser is created that uses dual decomposition. All these studies on Turkish dependency parsing used the first version [57] of the Turkish Treebank annotated in non-UD (non-Universal Dependencies) style.

The UD version of the Turkish Treebank, IMST, was evaluated using MaltParser in [75], however only a subset of the treebank was used by eliminating the non-projective dependencies in training and development sets. In our study, we included both projective and non-projective dependencies in IMST as the proportion of non-projective sentences in Turkish is too high to be ignored [6].

6.3. Methods

In order to improve the parsing performance of deep learning-based parsers, we design hybrid methods where the grammar-based information is fed into the deep network of a data-driven parser. We propose two different approaches for supplying the information extracted from the language grammar, the first one is via hand-crafted grammar rules for detecting dependency relations between words and the second one is by analyzing the underlying morphological structure of words.

We first give a brief description of the state-of-the-art neural parser used in this study in Section 6.3.1. We then explain our rule-based parsing method in Section 6.3.2 and show how these two methods are integrated to get a better parsing mechanism in Section 6.3.3. Finally, we describe our morphology-based enhancement method and its integration to the parser in Section 6.3.4.

6.3.1. Stanford’s Neural Dependency Parser

Stanford’s graph-based neural dependency parser [82] is the leading system in the CoNLL-17 shared task on UD parsing [73]. For the representation of input words, the model sums learned word embeddings, pretrained word embeddings, and character embeddings and then concatenate the resulting word vector with their corresponding part-of-speech (POS) tag embeddings. The parser includes three BiLSTM layers with 100-dimensional word and tag embeddings. It uses two biaffine classifiers: the arc classifier takes 400-dimensional head-dependent vectors produced by ReLU layers on top of the final BiLSTM layer to decide the head of a given token, and the label classifier uses 100-dimensional vectors to decide its label. For more information, see [82]. In this study, we use this parser as the baseline system.

6.3.2. A Rule-based Unlabeled Parsing Approach

We aim at enhancing the baseline parser by supplying linguistic information to the neural model. For this purpose, we design linguistically oriented rules for the dependency parsing of the Turkish language. These hand-crafted rules that are not completely free from false positives determine the head of the words in a sentence without assigning a label for the created dependency relation. Rather than applying them as a post-processing step that directly modifies the predictions made by the baseline parser, we integrate these rules to the parser via the dense representation of words in order to make them have an implicit effect on the decision of the parser (see Section 6.3.3).

Instead of a complete parser that creates a fully connected dependency graph, our rule-based system deals with the most difficult cases in predicting dependency relations in a sentence according to the parsing errors of the baseline system, such as complex predicates or multiple adverbs. The rules are created by considering the structural components of a sentence. We consider the relations between verbs, nouns, adverbs, and adjectives in a sentence as having the main importance and generate rules that deal

with these relations. The rules are based on the grammar rules extracted from [32].

Figure 6.1 shows the general mechanism of our rule-based parsing system. Our model takes a tokenized sentence as its input. Then the rules are applied in sequential order, considering the grammatical structure of Turkish. First, the ConsecAdv rule creates a list of the consecutive adverbs that are related to each other in the sentence, and sends this list to AdvAdj and AdvVerb rules as an input. Similarly, the ConsecAdj rule creates a list of the consecutive adjectives in the sentence that are related to each other and sends this list to the AdjNoun rule. After that, CmpxPred and NounComp rules are applied to the words of the sentence sequentially. Then, PossConst, AdvAdj, AdvVerb, AdjNoun, and NounVerb rules are applied to the remaining words in an iterative manner. This process continues until the heads of all the words in the sentence are associated with a dependency relation or no more dependency relations can be found in the sentence. The following subsections explain each rule in detail.

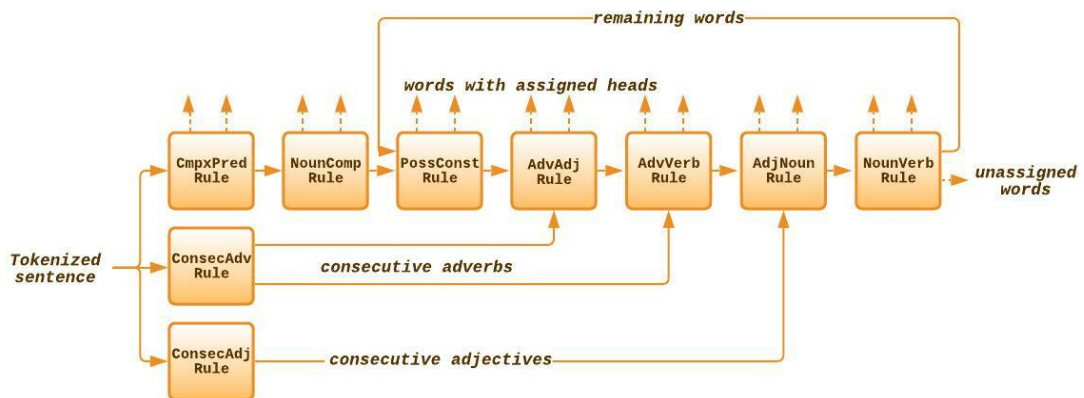
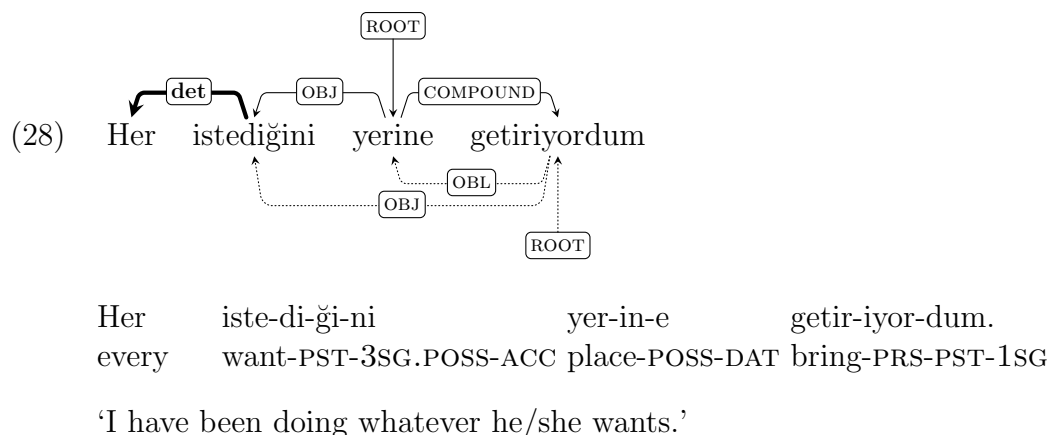


Figure 6.1. Our rule-based dependency parser.

Complex Predicates and Verbal Idioms (CmpxPred) Rule. This rule handles complex predicates (e.g., *kabul et* (accept), *sebep ol* (cause) etc.) and verbal idioms (e.g., *göz yum* (condone) etc.) in a sentence. Complex predicates in Turkish are made up of a bare nominal followed by one of free light verbs *ol*, *et*, *yap*, *gel*, *dur*, *kal*, *çık*, *düş*, *buyur*, *eyle* (page 143 in [32]). Yet, verbal idioms can have verbs in a wide range of words and the meaning of these verbs are changed when they are used in an idiom.

In complex predicates, head is the nominal part of the predicate. This is because light verb constructions in Turkish are not fully in parallel to their counterparts in languages like Persian where the light verb is considered as the source of all syntactic properties. In Turkish, the nominal part of the noun can still retain its argument structure and case-frame even within the absence of a light predicate as well observed in the literature [140–143]. Since the nouns not only make a semantic contribution but also determine the case and argument structure properties of the complex predicate, we take the nominal part of the complex predicate as the head of the construction. Yet, due to insufficient amount of training data, parsers usually fail to detect these multi-word predicates [144] and consider the verb of such predicates as the head of the relation. Example 28 shows such a false annotation done by the trained baseline deep learning-based parser.⁶ The parser falsely predicts that the verb *getir* (*bring*) is the **root** word and *yerine* (*to its place*) is an **oblique** of the **root** word. In fact, *yerine getir* (*fulfill*) is a verbal idiom and the verb *getir* is the verbal component of the complex predicate.



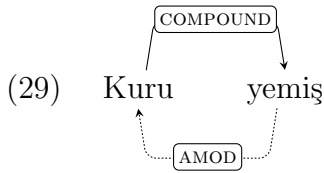
This rule correctly constructs the dependency relations between the words of such predicates. In order to detect such verbal compounds however, it needs a dictionary that lists complex predicates and idioms in Turkish. We collected approximately 8K complex predicates [14] using the Turkish Proverbs and Idioms Dictionary supplied by the Turkish Language Association (TDK) [145] and from various online Turkish resources. The CmpxPred rule searches for complex predicates and idioms in a sentence. When such a predicate is found, the second word of the predicate is set as a dependent of

⁶All examples in this chapter that include dotted lines (false annotations) are taken from the output of the baseline parser.

the first word and given **cmp** as the rule-encoding. Since the head of the second word is found, it is eliminated from the remaining words list.

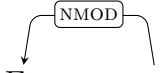
Noun Compounds (NounComp) Rule. In Turkish, there are two types of noun compounds: bare compounds that are treated as head-level (X^0) constructions and $-(s)I(n)$ compounds where the first noun has no suffixes while the second noun in the compound takes the 3rd person possessive suffix $-(s)I(n)$ (page 94 in [32]). In terms of dependency grammar representations, in (X^0) constructions the head word of the compound is the first noun, whereas in $-(s)I(n)$ compounds the first noun is dependent on the second noun. Note that, (X^0) constructions are not head-initial but considered as head-level constructions where there is no syntactic difference between words of the compound. In head-level compounds, two heads come together to form a new complex word. Syntax cannot treat head-level compounds as having a phrasal internal structure.

Differentiating between these two noun compound types is not easy for parsers in the absence of large amount of training data. Examples 29 and 30 show an (X^0) compound *kuru yemiř* (*dried nuts*) and a $-(s)I(n)$ compound *ev yemeęi* (*homemade food*), respectively. In Example 29, both of the words are in their bare forms with no suffixes and form a noun compound with the head word being *kuru* (*dried*). Yet, the parser falsely predicts *kuru* as an adjective modifier of *yemiř* (*nuts*).

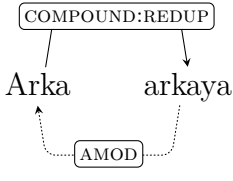


kuru yemiř
dry nut

‘Dried nuts’

- (30) 
 ev yemeği
 home food-3SG-POSS
 ‘Homemade food’

Reduplicated compounds are also handled by this rule. Reduplication is a common process in Turkish [90]. Reduplicated words construct reduplicated compounds. However, the parser sometimes cannot recognize this idiomatic structure and fails to construct the compound. Example 31 shows this kind of confusion where the parser falsely assigns the first word of the compound as an adjective modifier of the second.

- (31) 
 arka arkaya
 back back-DAT
 ‘Repeatedly’

To detect these cases, the NounComp rule utilizes large lexicons and detects the noun compounds in sentences with the help of these lexicons. We extracted three different lexicons from the official noun compounds dictionary of Turkish language published by the Turkish Language Association. These three lexicons are for noun compounds, possessive compounds, and reduplicated compounds with, respectively, the sizes of approximately 1.5K, 7K, and 2K entries [14]. We classified each entry in the dictionary as one of the three kinds of compounds according to their lexical classes. The NounComp rule searches through these lexicons and by this way detects noun compounds, possessive compounds, and reduplicated compounds in the sentences. In noun compounds and reduplicated compounds, the second word is set as a dependent of the first word, whereas in possessive compounds, the first word is set as a dependent of the second word. As rule-encodings, **com** is given to the dependent components of detected noun compounds, **fla** is given to the dependants of reduplicated compounds, and **nmo** is given to the dependent words in possessive compounds. The words whose heads are found are then eliminated from the remaining words list.

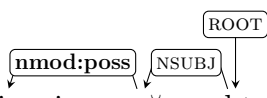
Possessive Construction (PossConst) Rule. The PossConst rule includes both *genitive-possessive constructions* and *possessive compounds* that cannot be detected by the NounComp rule. In genitive-possessive constructions, the first noun of a noun phrase takes a genitive suffix $-(n)In$ that represents the ownership of the first noun over the second noun. The second noun takes a possessive suffix $-(s)I(n)$ (page 161 in [32]). In possessive compounds, there is no genitive suffix and the first noun in the compound appears without any case marking (page 96 in [32]). Although it is easy for the parser to detect the genitive-possessive construction relations due to the existence of a genitive suffix, detecting possessive compounds is a challenging task. Because in possessive compounds, there does not exist a genitive suffix in the first noun and the possessive suffix $-(s)I(n)$ in the second noun is confusing since it appears the same as the accusative suffix $-(y)I$ when the suffix initial (s) is dropped in nouns ending with a consonant.

This situation is depicted in Examples 32, 33 and 34. The only difference in Examples 32 and 33 is that the genitive suffix showing the possession exists in 32 and is omitted in 33. In both sentences, the subject of the sentence is the word *yağ* (*oil*) and the two nouns form a possessive construction. However, this is not the case in Example 34. Here, the subject of the sentence is the word *makine* (*machine*), and the word *yağ* (*oil*) is the object of the verb *akit-* (*drip*). The confusion originates from the use of the same consecutive nouns, *makine yağ**ı*, in both of the example sentences 33 and 34. However, the *-ı* suffix of the word *yağ* in 34 is actually an accusative suffix and hence the two nouns in 34 do not form a compound. In order to help the parser to differentiate between these two cases in sentences, we construct the PossConst rule that identifies whether there is a compound relation between two consecutive nouns or not.

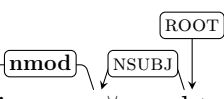
When two consecutive nouns are detected, the rule checks whether there is a genitive suffix in the first noun. If yes, it is set as a dependent of the second noun, given **nmo** as the rule-encoding, and dropped from the remaining words list. If the first noun is in bare form and the second noun has a possessive suffix, then the first noun

is set as the dependent on the second noun. As stated, the third person possessive suffix $-(s)I(n)$ can be confused with the accusative suffix $-(y)I$ when they are attached to a word ending in a consonant. In this case, both suffixes reduce to the form \imath , i , u , or \ddot{u} . To prevent this confusion, the rule analyzes the morphological features of the word and checks if it is identified as accusative (example 34) or not (example 33). If it is identified as the possessive suffix, the PossConst rule assigns the first noun as a dependent of the second noun, gives **nmo** rule-encoding, and the first noun is eliminated from the remaining words list.

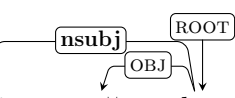
In addition, PossConst deals with multi-word proper nouns and determiner-noun relations. When the PossConst rule detects a multi-word proper noun, all the following consecutive proper nouns are set as dependent on the first proper noun, given **cop** as the rule-encoding, and dropped from the remaining words list. When it detects a noun that is preceded by a determiner, it sets the determiner as dependent on the noun. The dependent word is given **det** rule-encoding and dropped from the remaining words list.

- (32) 
Makinenin yağ-1 ak-t1.
machine-GEN oil-3SG-POSS leak-PST-3SG

‘The machine’s oil leaked.’

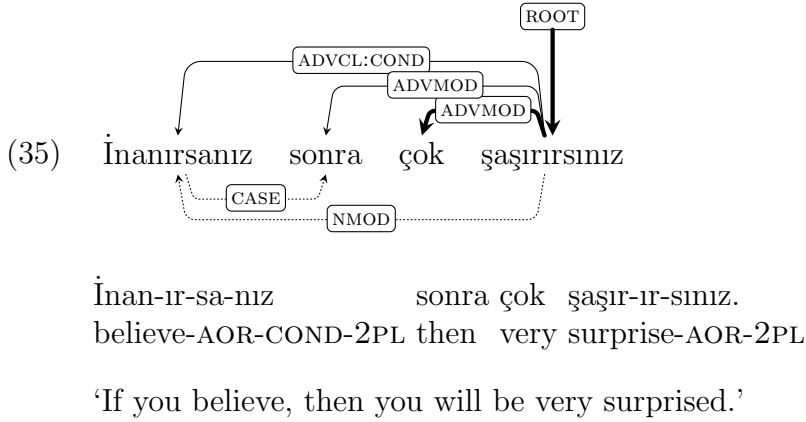
- (33) 
Makine yağ-1 ak-t1.
machine oil-3SG-POSS leak-PST-3SG

‘Machine oil leaked.’

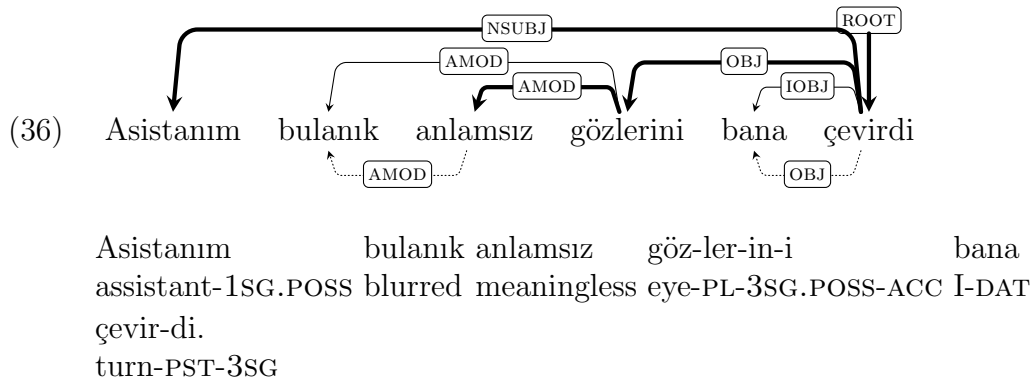
- (34) 
Makine yağ-1 akıt-t1.
machine oil-ACC drip-PST-3SG

‘The machine dripped the oil.’

Consecutive Adverb (ConsecAdv) Rule. Consecutive adverbs are also hard-to-detect with data-driven parsers. For instance, the first adverb *sonra* (*then*) and the second adverb *çok* (*very*) are both dependents of the verb *şaşırsınız* (*be surprised*) in Example 35. However, the parser falsely predicts the word *sonra* as the dependent of the previous word *inanırsanız* (*if you believe*) with **case** label. The ConsecAdv rule handles such consecutive adverbs in a sentence. We observe that, if there are two consecutive adverbs in a sentence, usually there are two cases: either the first adverb is dependent on the second adverb or they are both dependent on the same head word. So, when two consecutive adverbs are found, the method checks whether the first adverb belongs to the group of adverbs that emphasize the meaning of the next adverb or not (page 213 in [32]). This is done via searching through a list of adverbs of quantity or degree taken from [32] (pages 210-211). If yes, the first adverb is set as a dependent word of the second adverb, given **adv** as the rule-encoding, and dropped from the remaining words list. If not, these two adverbs are put in a list (which will be called *the consecutive adverbs list* throughout the article) and the first one is dropped from the list of remaining words. When the head word of the second adverb is found later, the first adverb is also bound to the same head word.



Consecutive Adjective (ConsecAdj) Rule. Consecutive adjectives are another troublesome word group which the parser sometimes fails to parse correctly. Example 36 shows such an annotation.

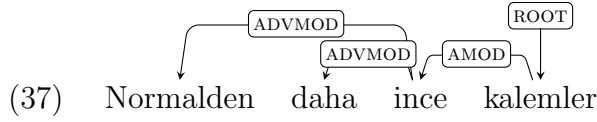


‘My assistant turned his/her blurred meaningless eyes towards me.’

The parser falsely considers the word *bulanık* (*blurred*) as an adjectival modifier of the word *anlamsız* and assigns **amod** to *bulanık*. In fact, it is an adjective describing the word *gözler* (*eyes*) and should be an **amod** dependent of the word *gözler*. The ConsecAdj rule is created to prevent this type of errors. This rule finds all the consecutive adjectives in a sentence. Usually, two consecutive adjectives are dependent on the same word. So, when two consecutive adjectives are found, these adjectives are put into a list (which will be called *the consecutive adjectives list* from now on) and the first one is dropped from the list of remaining words. When the head word of the second adjective is found later by the parser, the first adjective is also set as a dependent of the same head word.

Adverb-Adjective (AdvAdj) Rule. The AdvAdj rule handles adverb-adjective relations in a sentence. For every two consecutive words in the sentence, if the first word is an adverb and the second word is an adjective, and if the adverb is a quantity or degree adverb, then the adverb is set as a dependent of the adjective word (pages 175-180 in [32]) and given **adv** rule-encoding. When the head of an adverb is obtained in this way, the rule checks whether the adverb is in *the consecutive adverbs list* supplied by the ConsecAdv rule. If yes, the consecutive adverbs of this adverb are also set as dependents of the same head word and given **adv** as rule-encodings. So, in Example 37, when the AdvAdj rule detects the degree adverb *daha* (*more*) is followed by the adjective *ince* (*thin*), it sets *daha* as a dependent of *ince*. The rule then checks the *consecutive adverbs list* previously created by the ConsecAdv rule and finds that the adverb *normalden* (*than normal*) is a consecutive adverb of the adverb *daha*. So, the

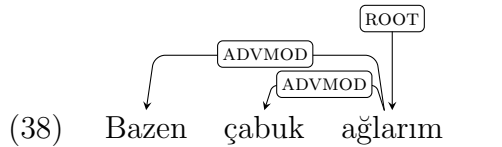
adverb *normalden* is also set as a dependent of the adjective *ince*.



Normal-den daha ince kalem-ler
 normal-ABL more thin pencil-PL

‘Pencils thinner than normal’

Adverb-Verb (AdvVerb) Rule. For every two consecutive words in a sentence, if the first word is an adverb and the second word is a verb, and if the adverb is not one of *bile* (*even*), *-DAn önce* (*before something*), *-DAn sonra* (*after something*) that modify the preceeding word, then the AdvVerb rule sets the adverb as a dependent of the verb (page 189 in [32]) as in the relation between the adverb *çabuk* (*quickly*) and the verb *ağlarım* (*I cry*) in Example 38. Otherwise, it sets the previous word of the adverb as its head. As the head of an adverb is found, the AdvVerb rule checks whether the adverb is in *the consecutive adverbs list* supplied by the ConsecAdv rule. If yes, the consecutive adverbs of this adverb are also set as dependents of the same head word and given **adv** rule-encoding.

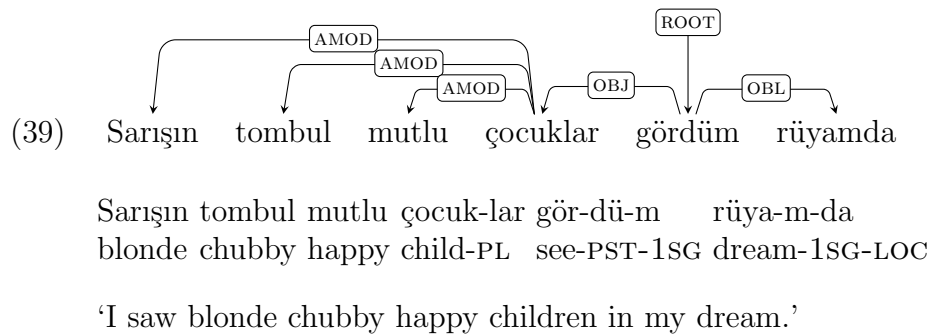


Bazen çabuk ağla-r-ım
 sometime quick cry-AOR-1SG

‘Sometimes I cry quickly.’

Adjective-Noun (AdjNoun) Rule. The AdjNoun rule constructs adjective-noun relations. For every two consecutive words in the sentence, if the first word is an adjective and the second word is a noun, then the adjective is set as a dependent word of the noun (page 170 in [32]) and **amo** is given as the rule-encoding. Like for the adverbs, when the head of an adjective is found, the algorithm checks whether the adjective is in *the consecutive adjectives list* supplied by the ConsecAdj rule. If

yes, the consecutive adjectives of this adjective are also set as dependents of the same head word and given **amo** rule-encoding. So, in the case of Example 39, the three consecutive adjectives *sarışın* (*blonde*), *tombul* (*chubby*), and *mutlu* (*happy*) are all set as dependents of the noun *çocuklar* (*children*) by the AdjNoun rule.



Noun-Verb (NounVerb) Rule. After complex predicates and noun, adverb, and adjective compounds are detected and eliminated from the sentence, the final NounVerb rule assigns any unassigned noun or pronoun followed by a verb as a dependent of that verb and gives **nov** rule-encoding.

A summary of the rules and their corresponding rule-encodings are depicted in Table 6.1. Application of each rule on an example sentence is depicted in Figure 6.2. In the example, the first rules that are applied to the sentence are the ConsecAdv and ConsecAdj rules. These rules prepare *the consecutive adverbs list* and *the consecutive adjectives list*, respectively. *The consecutive adverbs list* stores the consecutive adverbs that should be bound to the same head word. Similarly, *the consecutive adjectives list* stores the consecutive adjectives which should have the same head word. After that, the CmpxPred and NounComp rules are applied consecutively. The CmpxPred rule finds the complex predicates and idioms in the sentence using a large lexicon while the NounComp rule detects the noun compounds, possessive compounds, and reduplicated compounds that also exist in the pre-built lexicons.

Table 6.1. Summary of rules in the rule-based system with examples for each encoding. Bold words in the *Example* column are dependents with an assigned rule-encoding. Note that, the second case of the ConsecAdv rule and the ConsecAdj rule do not assign a rule-encoding, but create lists to be used by other rules in subsequent steps.

Rule	Grammatical Structures	Rule encoding	Example	
CmpxPred Rule	Complex predicates	cmp	<i>kabul</i> → <i>etmek</i>	Eng: to accept
NounComp Rule	Noun compounds	com	<i>kara</i> → <i>tahta</i>	Eng: blackboard
	Possessive compounds	nmo	<i>armut</i> ← <i>sapı</i>	Eng: pear stalk
	Reduplicated compounds	fla	<i>yamuk</i> → <i>yumuk</i>	Eng: crooked
PossConst Rule	Genitive possessive constructions	nmo	<i>armutun</i> ← <i>sapı</i>	Eng: stalk of pear
	Possessive compounds	nmo	<i>armut</i> ← <i>sapı</i>	Eng: pear stalk
	Determiner-noun relations	det	<i>bir</i> ← <i>yol</i>	Eng: a way
	Multiword expressions	cop	<i>zor</i> ← <i>dur</i>	Eng: it is hard
ConsecAdv Rule	Consecutive adverbs:			
	1) adverb emphasizing adverb 2) adverbs sharing heads	adv	<i>çok</i> ← <i>daha</i> <i>elbette böyle yaptım</i> [<i>elbette, böyle</i>]	Eng: much more Eng: course I did like this
ConsecAdj Rule	Consecutive adjectives sharing heads		<i>küçük eski masa</i> [<i>küçük, eski</i>]	Eng: small old table
AdvAdj Rule	Adverb-adjective relations	adv	<i>daha</i> ← <i>uzun</i>	Eng: longer
AdvVerb Rule	Adverb-verb relations	adv	<i>hemen</i> ← <i>geldi</i>	Eng: came immediately
AdjNoun Rule	Adjective-noun relations	amo	<i>uzun</i> ← <i>ağaçlar</i>	Eng: long trees
NounVerb Rule	Noun-verb relations	nov	<i>eve</i> ← <i>gitti</i>	Eng: went to home

As the operations of these one-time rules are completed, the PossConst, AdvAdj, AdvVerb, AdjNoun, and NounVerb rules are applied to the sentence in a loop until none of the rules can be applied anymore. As for the example sentence in Figure 6.2, only two words remained unassigned out of fifteen. The complete set of dependency relations extracted by the rule-based process is shown on the bottom of the figure. The last line in the figure shows the encoding of the rule applied to each word, which are then used by the dependency parser as will be explained in 6.3.3.

6.3.3. Integrating the Rule-based Approach with Stanford’s Graph-based Parsing Method

Our approach for combining the rule-based method with Stanford’s neural dependency parser is to embed the dependency parsing rule information to the dense representations of the words. The purpose of this approach is to give the parser an idea about finding the correct head of the corresponding word. The parser uses the dependent-head decisions made by the rule-based method in its learning phase and comes up with more accurate predictions about the syntactic annotation of sentences.

In our method, first the input sentences are pre-processed by the rule-based system. For each word in a sentence, the rules decide the head of the word if applicable, and then a three-letter encoding that denotes the rule action applied (`cmp` for `CmpXPred`, `amo` for `AdjNoun`, etc.) is assigned to that word. During this process, the rule-encoded words are dropped from the remaining words list in the sentence and the rule-based system continues its process in a recursive manner until the head word of all words in the sentence are found or no more rule can be applied. Note that, each word is affected by at most one rule. The rules are applied sequentially and their order of application is defined considering the grammatical structure of Turkish. For instance, the noun compounds that are dealt with by the `NounComp` rule must be detected in the sentence before the `PossConst` rule that also deals with nouns because the components of noun compounds cannot be separated from each other. Similarly the `AdvAdj` rule that deals with adverbs followed by an adjective must be applied before the `AdjNoun` rule that handles adjectives followed by a noun because otherwise the `AdjNoun` rule would assign a rule-encoding to the adjective and it would be dropped from the sentence and hence, the adverb-adjective relation in the sentence could not be detected. After this rule-based pre-processing step, the rule-encoding of each word is assigned a 100-dimensional embedding vector and concatenated to the embedding vector of that word. The rule embeddings are initialized randomly and trained together with the rest of the network. Figure 6.3 depicts this scheme.

So, in our model, the rule vector representation is concatenated with the vector representation of Dozat *et al.* [82]. The parser takes these word vector representations with the rule-based parsing decisions as input and learns the relations between the words in the training phase. In this way, we anticipate that the parser will benefit from the decisions of the rule-based system and arrive at a more accurate dependency tree for a given sentence.

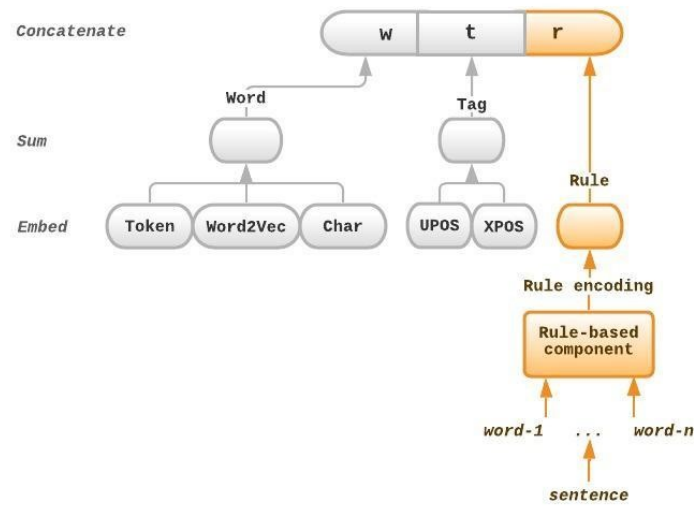


Figure 6.3. The word embedding representation of the hybrid model with the rule-based enhancement. As in the tag representation, rule-encodings are assigned randomly initialized embedding vectors at the beginning, which are then updated at each epoch during training.

6.3.4. A Morphology-based Enhancement for Dependency Parsing

In addition to the rule-based enhancement to the deep learning-based parser, we also propose to include the morphological information directly to the system. In this approach, we use morphemes of a word as an additional source. Our motivation relies on the fact that Turkish is a highly agglutinative language where the word structure is described by identifying the different categories of suffixes and determining which stems the suffixes may attach to and their orders. The suffixation process in Turkish

sometimes produces very long word forms that correspond to whole sentences in English [32]. The morphemes of a word hold important information in terms of the dependency relations that word belongs to. For instance, it was observed that the last inflectional morpheme of a word determines its role as a dependent in the sentence [57]. Based on such observations, we design three different methods to enhance the deep learning-based parser. The following subsections explain each model in detail.

6.3.4.1. The Inflectional Suffixes Model. In this model, all of the inflectional suffixes are extracted from the morphologically analyzed form of the word, embedded, and then concatenated to the vector representation of that word. The integration method is the same as in Section 6.3.3 in the sense that inflectional suffixes of each input word are represented with a 100-dimensional randomly initialized embedding vector which is then concatenated with the word and tag embeddings of the same word. Figure 6.4 depicts the creation of the dense representation of an example word *insanların* (*people's*).

6.3.4.2. The Last Suffix Model. Slightly different than the Inflectional Suffixes Model, here the same process is performed for only the last suffix of an input word. The vector representation of the last derivational or inflectional suffix of a word is added to the vector representation of that word. Figure 6.5 depicts the creation of the dense representation of the same example word in Figure 6.4, but this time using the Last Suffix Model.

6.3.4.3. The Suffix Vector Model. This model is a bit different than the previous two models. In the Suffix Vector Model, the input words are represented through a vector of all suffixes which the lemma of that word can and cannot take. The motivation behind this model comes from the idea that the role of a word form in a sentence can be determined by considering the suffixes that the lemma of that word never takes and the suffixes it frequently takes. For instance, inflectional suffixes in Turkish indicate how the constituents of a sentence are related to each other (page 65 in [32]).

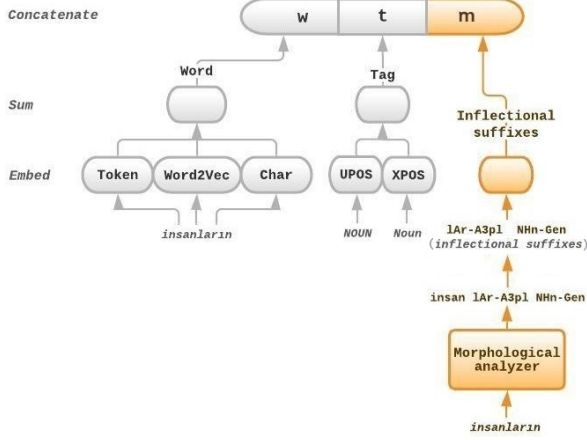


Figure 6.4. Dense representation of the word *insanların* (*people's*) using the Inflectional Suffixes Model.

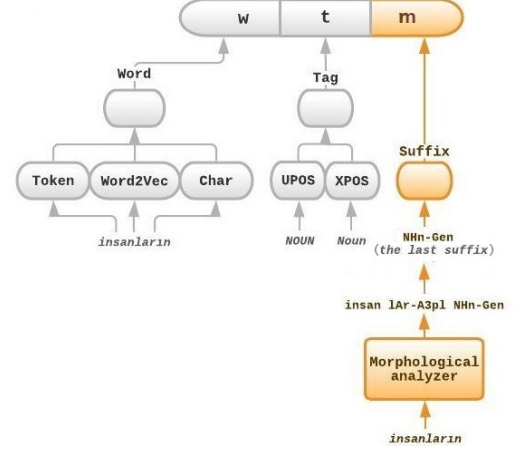


Figure 6.5. Dense representation of the word *insanların* (*people's*) using the Last Suffix Model.

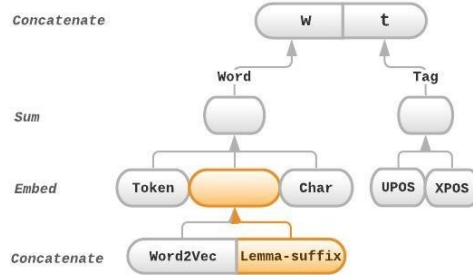


Figure 6.6. Word representation model of the system with the Suffix Vector Model.

For this purpose, we created a lemma-suffix matrix which consists of 40K unique lemmas and 81 inflectional and derivational suffixes in Turkish. Rows of the matrix list lemmas and columns show the normalized count of the times each lemma takes the corresponding suffix. To compute these statistics, we used the Newscor part of the Boun Web Corpus [85] which was created using news documents from three pioneering news portals in Turkish and includes 184M words. Morphological analyses of words in the corpus are predicted using the Turkish morphological analyzer and disambiguator of Sak *et al.* [85]. A small subset of the lemma-suffix matrix is shown in Figure 6.7.

This lemma-suffix matrix is then concatenated with the pretrained word embedding matrix used by the parser. For each word entry in the pretrained word embedding

matrix, the row vector in the lemma-suffix matrix that corresponds to the lemma of that word is found and this vector is concatenated to the end of the embedding vector of that word. Figure 6.6 depicts the word representation model of the system when we use the Suffix Vector Model.

	Ar	CA	CAsHnA	CH	CHK	DA	DAn	DH	DHk	DHkçA	DHr	HI	Hm	HmHz	Hn	HnHz	Hr
bakan[Noun]	0	2.50E-5	0	0	0	5.90E-5	0.002199	0	0	0	0.00016	0	0.000792	0.002089	1.70E-5	0.00011	0
bakanlık[Noun]	0	0.000227	0	0	0	0.001297	0.001441	0	0	0	4.20E-5	0	0.000211	0.000615	1.70E-5	0.000168	0
aynı[Verb]	0	0	0	0	0	0.000118	0.002738	0.014162	0.007313	1.70E-5	0.000522	0	0.000826	0.000379	7.60E-5	0.00016	0.005341
parti[Noun]	0	0.00016	0	2.50E-5	0	0.005552	0.006319	1.70E-5	1.70E-5	0	0.000876	0	0.000792	0.002275	5.90E-5	0.000413	1.70E-5
başkan[Noun]	0	0	0	0	0	7.60E-5	0.001281	0	0	0	0.000337	0	0.001045	0.002511	0	0.00016	0
duş[Verb]	0.010346	0	4.20E-5	0	0	0.001121	0.002578	0.031323	0.016167	0.000564	0.002637	0.003151	0.000379	0.000842	0.000143	0.000168	0.00182
uğra[Verb]	0	0	0	0	0	6.70E-5	0.000918	0.010876	0.010598	1.70E-5	0.000733	0.001533	0.000413	0.000388	0.00043	0.000126	0.002527
sonuç[Noun]	0	0	0	0	0	0.012721	0.001955	0.004322	0.001255	0	0.005451	0.001668	8.00E-6	5.90E-5	1.70E-5	0	0.001171
yaşam[Noun]	0	0	0	0	0	0.001719	0.001011	0	0	0	7.60E-5	0	0.001491	0.001508	8.00E-6	0.000556	0
yitir[Verb]	0	0	0	0	0	5.90E-5	0.00064	0.007001	0.00294	4.20E-5	0.000329	0.001062	6.70E-5	0.000564	0	5.10E-5	0.001188
zinc[Noun]	0	0	0	0	0	0.009309	0.001794	0	0	0	7.60E-5	0	0	2.50E-5	0	0	0
deplasman[Noun]	0	0	0	9.30E-5	0	0.012376	0.000354	0	0	0	2.50E-5	0	0	1.70E-5	0	0	0
devir[Verb]	0	0	0	0	0	5.90E-5	0.000211	0.003362	0.000657	0	3.40E-5	0	8.00E-6	5.10E-5	8.00E-6	1.70E-5	0.000388
rakip[Adj]	0	0	0	0	0	0.000404	0.000463	0	0	0	0.000404	0	0.000329	0.004111	8.00E-6	0.000447	0
ön[Noun]	0	0	0	0.000244	0	0.025333	0.000455	8.00E-6	0	0	0.000236	0	0.002763	0.051619	0	0.001289	0
oyun[Noun]	0	2.50E-5	8.00E-6	0.167443	8.00E-6	0.006967	0.011247	0	8.00E-6	0	0.001011	8.00E-6	0.003252	0.004575	0.000126	0.000227	0
risk[Noun]	0	0	0	0	0	8.40E-5	0.001196	0	0	0	0.000379	0	8.00E-6	8.40E-5	0	0.000126	0
et[Verb]	0.058771	8.00E-6	0.000548	8.00E-6	0	0.004945	0.014533	0.320438	0.1727	0.001171	0.052773	0.371433	0.010059	0.012401	0.001121	0.005451	0.037128

Figure 6.7. A small subset of the lemma-suffix matrix created from the Newscor part of the Boun Web Corpus.

6.4. Experiments

We evaluated the Stanford’s neural parser as our baseline and the proposed hybrid parser with rule-based and morphology-based enhancements on the IMST Treebank (UD version 2.3), the Turkish PUD Treebank [81], and the BOUN Treebank [11]. In all experiments, default set of parameters are used for the deep network that produces the parse trees. We use 100-dimensional word vectors, POS tag vectors, and rule embedding vectors. The 3-layer BiLSTM modules of the parser have hidden layer size of 400 on each side. The arc MLP layer of the parser is 400-dimensional and the label MLP layer is 100-dimensional. All dropout probabilities are 0.33. We use Adam optimizer [146] with a learning rate of 0.002, training the models for a maximum of 30,000 iterations and with early stopping criterion as 5,000 iterations without improvement.

We evaluated each of the proposed models on the IMST Treebank which is the most frequently used treebank in the literature. The training part of the IMST Tree-

bank has 3,685 annotated sentences and the development and test parts have 975 annotated sentences each. The PUD and BOUN treebanks were used in the second set of experiments where we measured the effect of increasing the size of the training data to the parsing models. We include both projective and non-projective dependencies.

As in the baseline approach [82], we used 100-dimensional Turkish word vectors from the CoNLL-17 pretrained word vectors [107]. In the evaluation of the dependency parsers, we used the word-based unlabeled attachment score (UAS) and the labeled attachment score (LAS) metrics.

In our system, both the rule-based and the morphology-based methods are dependent on a morphological analyzer and disambiguator tool. For this purpose, we used the Turkish morphological analyzer and disambiguator tool by Sak *et al.* [85]. This tool takes the whole sentence as input and analyzes and disambiguates the words with respect to their corresponding meanings in the sentence. This property is very useful for Turkish because there are many words in Turkish which have multiple morphological analyses that can be correctly disambiguated only by considering the context the word is in. The accuracy of the tool on a disambiguated Turkish corpus was reported as 96.45% in [85].

Although our proposed enhancement methods use a morphological analyzer and disambiguator tool and do not rely on the gold lemmas, gold POS tags, or gold morphological features, the baseline parser used in this study needs input sentences in CoNLL-U format where the sentence is segmented to tokens and pre-processing operations such as lemmatization and tagging are supplied for each token. Since our main aim is to improve the parser performance, we supplied gold tokenization and POS tags to the parser in the evaluation of the models in order to observe the pure effect of the proposed methods on parsing. When we compared our best model with the state-of-the-art parsers on parsing raw text, we utilized the Turku Neural Parser Pipeline [84], an end-to-end system for parsing from raw text, by replacing its default parser component with our parsing model.

6.5. Results

6.5.1. Ablation Study

We made an ablation study to see how each rule contributes to the overall performance of the rule-based parsing model. We started from the baseline where no rule is applied and then add the rules one by one to the model. At each step, we trained multiple models using different seeds for each setting and evaluated them on the development set of the IMST Treebank. The attachment scores shown in this section are the averages of the attachment scores of these multiple models.

Table 6.2 shows the rules the parser uses at each step of the ablation study. In Step 5, we added ConsecAdv and AdvAdj rules to the parser at the same time. The reason for including these rules together is that, the ConsecAdv rule finds the consecutive adverbs that will possibly share the same head word and these consecutive adverb pairs are given to the AdvAdj and AdvVerb rules as input. When a dependency relation is constructed between an adverb and an adjective by the AdvAdj rule or between an adverb and a verb by the AdvVerb rule, these rules look up the consecutive adverbs list sent by the ConsecAdv rule and assign the same head to their corresponding pairs.

Similarly, we introduced the ConsecAdj and AdjNoun rules to the parser at the same time in Step 7. Because, the ConsecAdj rule finds the consecutive adjective pairs that should be set as dependent words to the same head word, and the list of these consecutive adjectives is given to the AdjNoun rule as input. When the AdjNoun rule forms a dependency relation between an adjective and a noun, it searches through the consecutive adjectives list and assigns the same head noun to the corresponding pairs of that adjective.

Table 6.2. The ablation study steps for the rule-based parser.

Steps	Rules
Step 1	No rule
Step 2	CmpxPred
Step 3	CmpxPred + NounComp
Step 4	CmpxPred + NounComp + PossConst
Step 5	CmpxPred + NounComp + PossConst + ConsecAdv + AdvAdj
Step 6	CmpxPred + NounComp + PossConst + ConsecAdv + AdvAdj + AdvVerb
Step 7	CmpxPred + NounComp + PossConst + ConsecAdv + AdvAdj + AdvVerb + ConsecAdj + AdjNoun
Step 8	CmpxPred + NounComp + PossConst + ConsecAdv + AdvAdj + AdvVerb + ConsecAdj + AdjNoun + NounVerb

Figure 6.8 shows effect of each rule to parsing performance in terms of attachment scores. We observe that, all rules except the AdvVerb and NounVerb rules improve the parsing performance whereas AdvVerb and NounVerb rules cause a drop in both UAS and LAS. A possible reason for this might be the over-generalizing structures of these rules. Considering the high frequency of complex sentences in Turkish which include one or more subordinate clauses in addition to the main clause, the risk of assigning the wrong verb as the head of an adverb is high for the AdvVerb rule. Similarly in the case of the NounVerb rule, there is a high probability of constructing a relation between a noun and a verb falsely when there are multiple verbs in a sentence.

So, we removed the AdvVerb and NounVerb rules from the rule-based parser and performed the ablation study again with the new setting. The effect of the rules to the performance is depicted in Figure 6.9. We observe that each rule now improves the parsing scores which means that none of the rules blocks the other and each of them contributes to the parsing performance of the system. All of the experiments on the rule-based parser were performed using this final configuration of the rules.

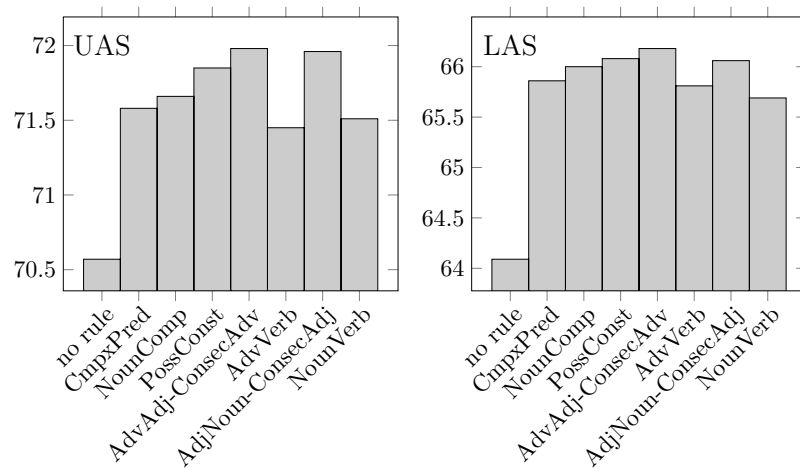


Figure 6.8. The effect of each rule to the parsing performance on the development set of the IMST Treebank. Each rule is added on top of the previous rules. So, in the first step with the label *no rule*, there is no rule used in the model. In the second step, the CmpxPred rule is added to the model. In the third step, the NounComp rule is added to the model which means both the CmpxPred and NounComp rules are present in the model. The integration of the other rules proceeds in the same manner.

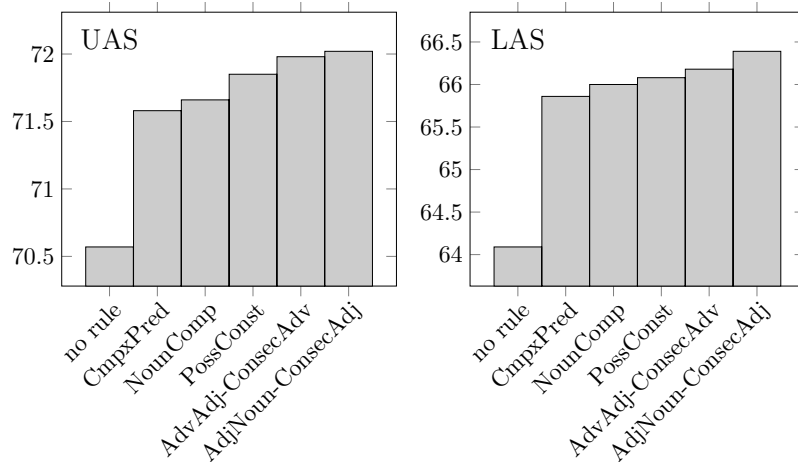


Figure 6.9. The effect of each rule to the parsing performance on the development set of the IMST Treebank, when the AdvVerb rule and the NounVerb rule are removed from the rule-based parsing system. Each rule is added on top of the previous rules.

6.5.2. Comparison of the Models

Table 6.3 shows the unlabeled and labeled attachment scores of the baseline system, our proposed enhancement models, and a state-of-the-art multilingual BERT-based parsing system Udify [147] on the test set of the IMST Treebank. For each setting, the average and standard deviation across five runs are reported.

We have five different hybrid models that are built on top of the baseline. Our first hybrid model use the proposed rule-based approach in Section 6.3.2. The second, third, and fourth hybrid models are the ones where we apply the corresponding versions of the morphology-based enhancement methods in Section 6.3.4. The last hybrid model is the combination of rule-based and morphology-based approaches. We selected the Last Suffix Model for this combination since it is the best performing one among morphology-based methods. We combine these models by concatenating their corresponding embedding vectors to the original input word vector representation.

Table 6.3. Attachment scores of the baseline parser, proposed models, and a state-of-the-art multilingual BERT-based parsing model (Udify) on IMST.

Parsing models	IMST	
	UAS	LAS
Baseline [82]	72.14 \pm 0.4	66.12 \pm 0.3
Hybrid - rule	74.03 \pm 0.3	67.99 \pm 0.1
Hybrid - inflectional suffixes	73.57 \pm 0.3	67.81 \pm 0.2
Hybrid - last suffix	73.95 \pm 0.1	68.25 \pm 0.2
Hybrid - suffix vector	73.09 \pm 0.3	66.96 \pm 0.3
Hybrid - rule and last suffix	74.37\pm0.4	68.63\pm0.4
Udify [147]	74.32 \pm 0.2	67.35 \pm 0.3

Experimental results show that all of our hybrid models outperform the baseline parser on the IMST Treebank with p-values lower than 0.01 according to the performed

randomization tests. We observe that the best performing model is the combination of the rule-based model and the Last Suffix Model with more than 2 and 2.5 points differences in, respectively, UAS and LAS scores when compared with the baseline.

We see that, among the three morphology-based models, best performing one is the Last Suffix Model. The success of this model matches with the observation that the last suffix of a word determines its role in a sentence [122]. This result suggests that the Last Suffix Model can accurately group the words using the last morpheme information for dependency parsing. Both UAS and LAS differences between this model and the other two morphology-based models are found to be statistically significant on the performed randomization test. From these results, we can conclude that the Last Suffix Model can be preferred over the other two morphology-based models with respect to the parsing performance and model simplicity. Although it outperforms the baseline on both scores, the Suffix Vector Model is the worst performing one among the proposed methods. Its relatively low performance can be attributed to its complex structure which includes all of the unique suffixes in Turkish. Filtering some of the suffixes by putting a frequency threshold during the construction of the lemma-suffix matrix and lowering the dimension of the vectors might improve its performance.

The performance of the rule-based model significantly outperforms the Suffix Vector Model on both UAS and LAS scores. When compared with the Inflectional Suffixes Model, the rule-based model outperforms the Inflectional Suffixes Model significantly on UAS score. However, its LAS score is only slightly better than the Inflectional Suffixes Model which leads the parsing accuracy difference to be insignificant in terms of LAS score. The rule-based model performs slightly worse than the Last Suffix Model according to the LAS score and slightly better than the Last Suffix Model according to the UAS score. Both differences are small and the performed randomization test results show that both of the differences are insignificant.

Yet, the best performance is reached when we combine the rule-based model with the Last Suffix model. The combined model outperforms all of the other models

and the performance differences are found to be statistically significant. This result suggests that rule-based and morphology-based approaches improve different aspects of the deep learning-based parser on the dependency parsing of Turkish. Actually we observe that, while the rule-based model is more successful in establishing dependency relations between words, the Last Suffix Model is better at determining relation types.

We also compared our models with the current state-of-the-art parsing system Udify, a transformer-based multilingual multi-task model that reached or exceeded state-of-the-art performance on many languages [147]. It utilizes multilingual BERT as its language model. We fine-tuned the multilingual Udify model on the training set of the IMST Treebank and evaluated on the test set of the IMST Treebank using gold segmentation and tokenization. We observe that our best hybrid model outperforms Udify significantly on LAS and performs slightly better than Udify on UAS.

All experiments were repeated five times. The p-values of model comparisons were obtained by performing the approximate randomization test [148] as described in [149] on the model outputs. We set the number of shuffles to 10,000.

Table 6.4. Comparison of our best hybrid model on the IMST test set with the top performing parsing systems in CoNLL 2018 shared task on multilingual parsing from raw text to Universal Dependencies.

Parsing models	IMST		
	LAS	MLAS	BLEX
Hybrid	65.06	55.94	60.69
HIT-SCIR	66.44	53.81	56.72
TurkuNLP	64.79	55.73	60.13
UDPipe Future	63.07	54.02	56.69
ICS PAS	63.54	52.51	58.89
CEA LIST	63.78	55.00	54.29

6.5.3. Parsing Performance on Raw Text

We also measured the parsing performance of our best hybrid model when the task is parsing from raw text where there is no gold segmentation or tokenization available. As in every other parsing system, the performance of our model was also affected negatively by the usage of automatic segmentation, tokenization, and tagging instead of the gold ones. Table 6.4 shows the comparison of our best parsing model with the state-of-the-art on parsing raw text. The systems in the table are the five best performing parsers in the CoNLL 18 Shared Task on Multilingual Dependency Parsing from Raw Text to Universal Dependencies [112]. The shared task used LAS, MLAS, and BLEX metrics to sort the participating systems.

In terms of LAS, our best model (using both rule-encodings and last-suffix information) is ranked second among the top five best systems participated in the shared task. The best performing system is HIT-SCIR [127] which incorporates an ensemble of three instances of Stanford’s neural parser [82] trained with different initializations and contextual word embeddings. Although HIT-SCIR is ranked first in LAS, our model outperforms it and is ranked first in MLAS and BLEX metrics. Our model also outperforms TurkuNLP, which again uses Stanford’s neural parser as the parsing component, in all three metrics.

Effect of Rule-based and Morphology-based Enhancements on Parsing. In Table 6.5, we depict the results of an analysis made on the proposed hybrid method to see how rules and morphology are affecting the parsing performance and what percentage of the input tokens are covered with these methods. We performed this analysis on the test set of the IMST Treebank and we excluded the punctuation from the analysis. We observe that the rules are covering 36.07% of the total tokens in the test set whereas the last-suffix information is included in 52.38% of the tokens. We further analyzed the tokens that get a rule-encoding to see how much different rules contributed to this amount. We counted the tokens encoded by the NounComp rule and the PossConst rule together because the areas of operation of these rules overlap and sometimes

the same rule-encoding is used by both rules. We did the same thing for AdvAdj-ConsecAdv and AdjNoun-ConsecAdj rule pairs too. From the statistics in the first part of Table 6.5, we see that 62.07% of the rule-encodings are resulted from the NounComp-PossConst rule pair. The AdjNoun-ConsecAdj rule pair follows it with 19.30%. The AdvAdj-ConsecAdv rule pair has 10.18% of the rule-encodings and the remaining 8.05% rule-encodings is resulted from the CmpxPred rule.

In the second part of Table 6.5, the performance of the hybrid and baseline parsers on the tokens for which the last-suffix information is available and on the tokens which are assigned a rule-encoding is given. On 4,295 tokens with last-suffix information, the hybrid parser made approximately 3.5 points and 4.5 points improvement over the baseline parser in UAS and LAS, respectively. On 2,958 tokens with rule-encodings, the hybrid parser outperforms the baseline parser by almost 3 points in both scores. Both performance differences being greater than the performance differences between the hybrid and baseline parsers on the whole test set (Table 6.3) signals the contribution of the two enhancements. It also suggests that increasing the coverage of the rules and morphology is likely to result in better parsing scores. When we compare the two parsers on individual rules, we see that the biggest effect is caused by the CmpxPred rule with adding almost 6 points to UAS and 5.5 points to LAS over the performance of the baseline. The AdjNoun-ConsecAdj rule pair improves the scores by 3 points, the effect of the NounComp-PossConst rule pair is 2.5 points in both scores. The AdvAdj-ConsecAdv has the lowest improvement rates with only a slight difference on UAS and almost 2 points increase in LAS.

6.5.4. Error Analysis on the Rules

To measure the impact of each rule on the parsing decision more explicitly, we form the following two questions: (i) For a given rule, of all the words that are given the correct rule-encoding by that rule, how many of them are predicted correctly (true positive: TP) by the parser and how many are missed (false negative: FN)? (ii) For a given rule, of all the words that are given a wrong rule-encoding by that rule, how

many of them are predicted falsely (false positive: FP) by the parser and how many are predicted correctly (true negative: TN) in spite of the misleading rule-encoding?

Table 6.5. Analysis of the proposed methods on the IMST test set.

Total token count	Tokens with last-suffix	Tokens with rule-encodings					
8,199	4,295 52.38%	2,958 36.07%	{	NounComp	AdvAdj	AdjNoun	
				&	&	&	
				CmpxPred	PossConst	ConsecAdv	ConsecAdj
				count	count	count	count
				250	1,836	301	571
				8.05%	62.07%	10.18%	19.30%
Performance of the parsers:							
	on tokens with last-suffix (4,295 tokens)	on tokens with rule-encodings (2,958 tokens)					
Performance of the hybrid parser:							
UAS:	73.04	78.23	{	72.40	78.27	82.40	78.46
LAS:	65.66	69.47		60.80	68.95	80.40	69.18
Performance of the baseline parser:							
UAS:	69.61	75.42	{	64.80	75.70	82.05	75.65
LAS:	61.14	66.83		55.20	66.67	78.73	66.19

To answer these questions, we created confusion matrices from the outputs of the hybrid parser which employs only the rule enhancement (hybrid - rule) and the baseline parser. Figure 6.10 depicts these matrices for the CmpxPred rule and NounComp-PossConst, AdvAdj-ConsecAdv, and AdjNoun-ConsecAdj rule-pairs. We observe that the existence of a correct rule-encoding helps the hybrid parser to make the right parsing decision (TP) and reduces the FN rates, (i.e., setting up the wrong dependency relation) when compared to the baseline parser. On the other hand, if the word is given a wrong rule-encoding, this time we observe that the hybrid parser has a bias towards this rule-encoding which results in a higher FP rate and a lower TN rate for the case of CmpxPred rule and the NounComp-PossConst rule pair. However, this negative effect is small compared to the positive effect of the rule-based enhancement on the

hybrid parser. We observe that even if the rules are not 100% correct individually, our proposed approach of incorporating the knowledge obtained from them into the neural parser’s learning has been successful in dependency parsing of Turkish.

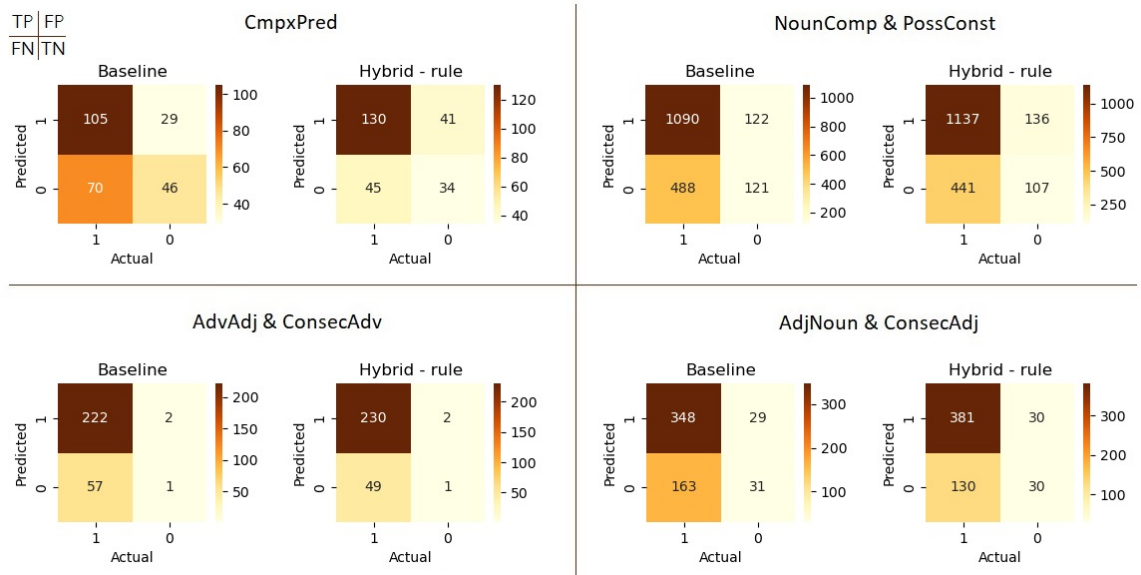


Figure 6.10. Confusion matrices for the case of detecting complex predicates (upper-left corner), noun compounds and possessive constructions (upper-right corner), adverb-adjective relations (lower-left corner), and adjective-noun relations (lower-right corner). Label 1 on axis *Actual* means the rule-encoding assigned is correct (e.g., the word is assigned the rule-encoding **cmp** by the CmpxPred rule and the word *is* a part of a complex predicate) and Label 0 on axis *Actual* means the rule-encoding assigned is wrong (e.g., the word is assigned the rule-encoding **cmp** but the word is *not* a part of a complex predicate). Label 1 on axis *Predicted* means the parser predicts the relation of the word in accordance with its rule-encoding (e.g., the word is assigned the rule-encoding **cmp** and the parser predicts it as a part of a complex predicate) and Label 0 on axis *Predicted* means the parser predicts the relation of the word by conflicting its rule-encoding (e.g., the word is assigned the rule-encoding **cmp** but the parser does *not* predict it as a part of a complex predicate).

We conclude that the proposed methods increase the parsing accuracy of Turkish which has insufficient amount of training data. The aim of our approach is to

give the parser additional information in constructing the dependency relations when learning from the training data is inadequate, i.e. there is not sufficient data to learn specific relations. The results show that both the rule-based and morphology-based enhancements on the neural parser improve the parsing accuracy significantly. The experiments performed on Turkish suggest that the languages with rich morphology need language-specific treatments and remarkably benefit from the usage of the basic grammar rules as well as from the inclusion of morphological suffix information.

6.5.5. The Effect of Training Data Size

After the experiments made on the IMST Treebank for measuring the performance of each proposed model, we made additional experiments in a larger setup to understand how the improvement gained by the hybrid approach changes with different amounts of training data.

The training set of the IMST Treebank consists of 3,685 sentences. To be able to observe the effect of the gradual increase of the training data more accurately, we additionally used the BOUN Treebank [11], a newly introduced Turkish treebank annotated in UD style. Being the largest dependency treebank in Turkish, the BOUN Treebank includes a total of 9,761 manually annotated sentences (7,803 training, 979 development, and 979 test sentences) from various topics including biographical texts, national newspapers, instructional texts, popular culture articles, and essays. The source texts were taken from the Turkish National Corpus (TNC) [61]. Decisions regarding the annotation of the BOUN Treebank were made in line with the recent efforts for unifying the Turkish UD treebanks through manual re-annotation [81]. In this context, the IMST Treebank and Turkish PUD Treebank were also re-annotated manually [80, 81] and these re-annotated versions comply with the BOUN Treebank in terms of annotation decisions. Although the current version of the PUD Treebank (which is also the version used in our work) is this re-annotated version, the re-annotated version of the IMST Treebank has not been validated. So, we use the original version of the IMST Treebank in all our experiments. However, note that there are some major dif-

ferences in the annotations of the current version of the IMST Treebank and the BOUN Treebank. For more detailed information, see [80] and [11].

For this second set of experiments, the training set of the IMST Treebank was split into 7 batches of 500 sentences (the last batch includes 685 sentences), and the training set of the BOUN Treebank was split into 8 batches of 1000 sentences (the last batch includes 803 sentences). Then the training sets used in the experiment were created by first adding the 500-sentence batches of IMST on top of each other one by one and then continuing the process with the 1000-sentence batches of the BOUN Treebank.

In each setup, the trained models were evaluated on four different test sets. These test sets are the test set of the IMST Treebank, the test set of the BOUN Treebank, the Turkish PUD Treebank, and the combined set of these three sets.

Figure 6.11 depicts the results of these experiments. The four plots in the first column show the UAS performance of the proposed hybrid model and the baseline model on the four test sets and the plots in the second column demonstrate the LAS performance of the models. The vertical dashed line in the plots shows the point where additional BOUN training sentences are beginning to be added on top of the training set of the IMST Treebank.

From these performance curves in the figure, we observed the following:

- The proposed hybrid model outperforms the baseline model consistently across all the test sets at every step.
- Adding additional training sentences from the BOUN Treebank has a positive effect on the parsing performance on all test sets except the test set of the IMST Treebank. The fluctuating performance change on the IMST test set results from the annotation difference between the IMST Treebank and the BOUN Treebank.
- The performance of the baseline model on the PUD test set does not always

increase when there are only IMST sentences in the training set. When we start to add sentences from the BOUN Treebank, the performance is first decreased and then started to increase again as more BOUN sentences are included in the training set. This might again stem from the conflicting annotation differences between IMST and BOUN treebanks.

- Across the BOUN, IMST, and PUD test sets, the performance gap between the hybrid and the baseline parsers is 4.3 points in UAS and 4.8 points in LAS on average when training data consists of 500 sentences. The size of this gap decreases to 2.2 points in UAS and 2.6 points in LAS when training size is 11,488 sentences.

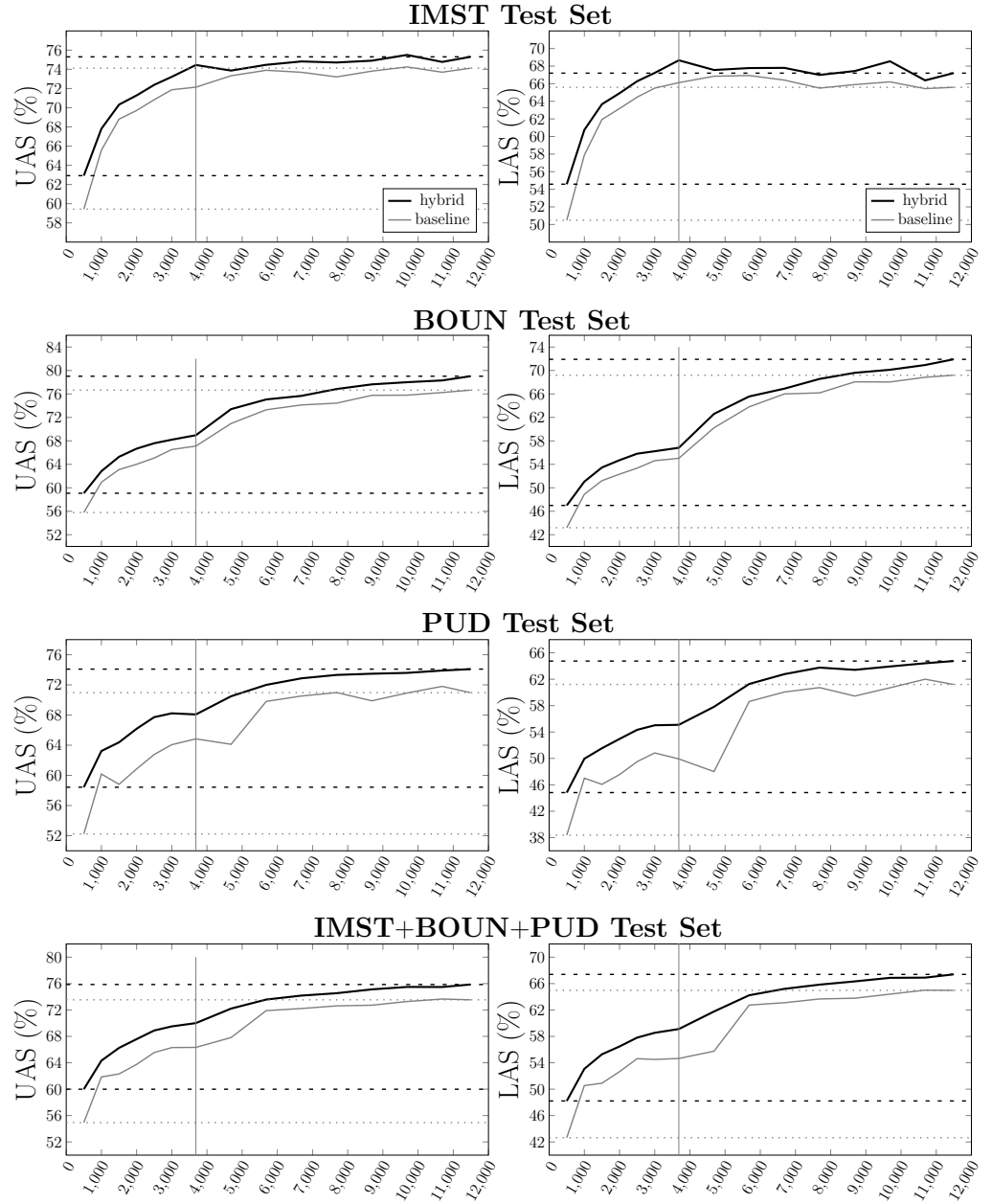


Figure 6.11. Effect of increasing the training data size on parsing performance of our hybrid model and the baseline model [82] in terms of UAS and LAS. The vertical line in the plots denotes the point where additional BOUN sentences are beginning to be added to the training data. The horizontal lines show the performance differences between two parsers at the beginning and end of this incremental process (the black horizontal dashed line is for the hybrid parser and the gray horizontal dotted line is for the baseline parser).

6.6. Generalization to Other Languages

Although this study focuses on the improvement of Turkish dependency parsing, the proposed models can be adapted to other languages as well. Below we first state the adaptation of the rule-based component, which is followed by the adaptation of the morphology-based component.

6.6.1. Adapting the Rule-based Parsing Approach

The ConsecAdj, AdvVerb, AdjNoun, and NounVerb rules can be directly applied to any language as long as the underlying structures exist in the language. The ConsecAdv and AdvAdj rules use adverb lists to make decisions. They can be applied to any language by supplying the adverbs specified in the rule descriptions for that language.

The PossConst rule needs a small adaptation to work for other languages because it uses the genitive and possessive marks when constructing possessive compounds. The genitive and possessive marks should be changed with the corresponding ones of the language, if applicable.

For the CmpxPred rule which handles complex predicates, we use a dictionary of complex predicates and idioms for Turkish. If there is a similar structure in the language the model is adapted to, supplying such a dictionary to the CmpxPred rule will be sufficient. Similarly, providing the three lexicons that separate between noun compounds, possessive compounds, and reduplicated compounds for the target language will be sufficient to adapt the NounComp rule.

We stated above how the rules proposed in this study can be adapted to other languages. In addition to such rule adaptations, and more importantly, effective rules for any language can be found by making a manual error analysis on the parser outputs. The grammar rules that hold for the relations between verbs, nouns, adverbs,

and adjectives in a sentence can easily be applied for that language in general. The rule applications can then be integrated with the dependency parser by following the proposed strategy explained in Section 6.3.3.

6.6.2. Adapting the Morphology-based Enhancement Approach

To create our suffix-based models, we utilize a morphological analysis tool for Turkish. A similar approach can be followed for any target language with agglutinative morphology. All of the three morphology-based models can be adapted by extracting the derivational and inflectional affixes in the target language. Moreover, other suffix-based models (e.g. using particular suffixes rather than the last suffix) can be employed depending on the specifics of the language by using the strategy proposed in Section 6.3.4.

6.7. Conclusion

In this chapter, we introduced a new rule-based method and three morphology-based methods for improving the accuracy of a deep learning-based parser on Turkish dependency parsing. In the rule-based approach, decisions made by the rule-based system are integrated into the word representation model of the deep learning-based parser. In the morphology-based approach, we experimented with the morphemes of the words by investigating different methods to integrate the information extracted from the morphemes into the word vector representation model of the parser. We observed that the best method of utilizing morphological information in terms of the dependency parsing of Turkish is using the last suffix of a word. A combination of the rule-based and morphology-based approaches outperforms all of the other proposed models as well as the baseline system, suggesting that Turkish dependency parsing benefits both from the linguistic grammar rules and the additional morphological information extracted from the input words. The experimental results show that our enhancement methods are useful for purely deep learning-based parsers, especially when there is not sufficient amount of training data to learn the dependency relations.

The results also indicate that the best performing model of the proposed approaches outperforms the state-of-the-art on parsing of the IMST Treebank. The source code of the hybrid parser is available at [14].

7. A SEMI-SUPERVISED DEPENDENCY PARSER FOR LOW-RESOURCE LANGUAGES: A CASE STUDY FOR CODE-SWITCHED TEXTS

7.1. Introduction

In this chapter, we expanded the focus of this thesis by proposing a novel parsing method with semi-supervised enhancement for low-resource languages and applied the proposed method on low-resource code-switching (CS) language pairs as a case study.

Code-switching is the creation of utterances by combining phrases and word forms from multiple languages. Although much work has been done on the syntactic parsing of monolingual languages, CS language pairs are quite understudied in this regard. There have been only a few studies on CS dependency parsing [150–152], each focusing only on a single CS language pair. Although CS dependency parsing also benefited from the recent rise of multilingual and cross-lingual NLP models as shown by van der Goot *et al.* [153], these models, which are usually trained on monolingual corpora, are insufficient on CS parsing. The poor performance on CS language pairs is not only due to the lack or scarcity of the training data but also because of the shortage on resources required by deep neural models such as pretrained embeddings, language models, or even raw data. In addition, each language composing a CS language pair inherits its own structural difficulties which contributes a good deal to the problem.

Recently, a small number of CS treebanks were manually annotated within Universal Dependencies (UD) [4]. Even though these treebanks have little to no training data, their existence provides an opportunity to study dependency parsing also on CS language pairs.

In such low-resource scenarios, utilizing raw data can be helpful in boosting the performance. A common method to benefit from raw data is self-training [154], a semi-

supervised approach where a small number of labeled data is used to train a model that is later used to predict labels for unlabeled data. This pseudo-labeled data is then combined with the initial data to re-train the model. This method is usually found successful in low-resource scenarios [155, 156], although error propagation is a known problem when pseudo-labels are noisy.

With very restricted resources, we hypothesize that CS dependency parsing can also benefit from unlabeled data. Based on this hypothesis, we ask the following question: is using pseudo-labeled data directly beneficial for CS dependency parsing or can we find better ways of integrating the knowledge from pseudo-labeled data?

Starting from this question, we follow a deep contextualized self-training approach [157] and integrate semi-supervised auxiliary tasks to the parsing architecture to enhance CS dependency parsing. Our method enhances a widely-used BiLSTM-based parser [117] by training parsing-related auxiliary sequence labeling tasks on automatically labeled data and combining these trained auxiliary task models with the base parser through a gating mechanism. We introduce new sequence labeling tasks that are shown to be beneficial in improving the parsing performance. Seeing the success of our semi-supervised enhancement method on the BiLSTM-based parser, we ask a second question: can we reach even better parsing scores if we combine this enhancement method with XLM-R [49], a state-of-the-art (SOTA) transformer-based language model that shows superior performance on many NLP tasks? Our experimental results demonstrate notable success of our proposed models over the previous state-of-the-art on CS UD treebanks. Our contributions are as follows:

- We employ a semi-supervised learning approach based on auxiliary tasks for CS dependency parsing. We present the first study with a focus on parsing all CS UD treebanks and achieve SOTA results on all of them.
- We introduce novel sequence labeling tasks including a CS-specific one, that capture syntactic information better and hence improve dependency parsing.
- We adapt this method to the powerful XLM-R model and elaborate the effective-

ness of this approach when combined with XLM-R-based word representation for dependency parsing. We demonstrate that the mighty transformer model remains inadequate for the case of low-resource CS parsing.

- We apply the proposed approaches to the parsing of Turkish. Results of the experiments performed on the IMST Treebank show that the proposed approach is also effective for Turkish for which the data scarcity problem is not as severe as for CS language pairs.

This chapter is based on the collaborative work with the Institute for Natural Language Processing at University of Stuttgart published as [10] and supported by the German Research Foundation (DFG) via project CE 326/1-1 “Computational Structural Analysis of German-Turkish Code-Switching”.

7.2. Related Work

Code-switching dependency parsing is a newly-studied research area. The first CS UD treebank was created by Bhat *et al.* [150] which included only a test set of Hindi-English sentences. In the absence of CS training data, the test set was split to monolingual fragments and existing Hindi and English monolingual treebanks in UD were used to parse these fragments. Bhat *et al.* [158] extended this dataset with a CS training set. They trained a BiLSTM architecture on this additional training data by also integrating syntactic knowledge extracted from monolingual treebanks.

Partanen *et al.* [151] laid the first foundations of a Komi-Russian UD treebank with 25 CS sentences. They adopted a multilingual parsing approach [159] and used Russian and Komi monolingual training data with bilingual Komi-Russian word embeddings. Later, this treebank expanded into the Komi-Zyrian IKDP treebank [160].

Çetinoğlu and Çöltekin [161] created a Turkish-German UD treebank from a Turkish-German spoken corpus. Seddah *et al.* [162] introduced the Maghrebi Arabic-French treebank and performed parsing experiments on the treebank using UDPipe

[163]. This treebank is yet to be included in the UD. A Frisian-Dutch UD treebank which includes only test data was introduced by Braggaar and van der Goot [152]. The authors performed data selection from eight related monolingual treebanks using Latent Dirichlet Allocation [164] to create a training set. Their experiments performed using a deep biaffine parser [153] demonstrated no significant performance difference between training the parser on the selected training set and only on a Dutch monolingual treebank.

Lately, multilingual and cross-lingual parsing studies have begun to include CS treebanks in their experimental setups. Van der Goot *et al.* [153] presented a multi-task learning tool that utilizes multilingual BERT [44] to perform several NLP tasks, including dependency parsing. Evaluation was done on all available UD treebanks which include CS UD treebanks mentioned above. The model was fine-tuned on training set of each treebank, which is also the case for Hindi-English and Turkish-German CS treebanks. For Frisian-Dutch and Komi-Russian CS treebanks with no training data, they used Dutch Alpino and Russian SynTagRus treebanks, respectively. Müller-Eberstein *et al.* [165] applied a sentence level genre-based data selection from UD treebanks in a cross-lingual setup. They trained a multilingual BERT-based biaffine parser [153] for 12 low-resource UD treebanks including Hindi-English and Turkish-German CS treebanks.

Our study on CS dependency parsing differs from the previous work in the sense that none of the previous work utilized raw CS data to improve parsing in a semi-supervised scheme.

7.3. Methods

7.3.1. Base Parsing Model

Our base parser is a neural graph-based parser by Dozat *et al.* [117] that uses two biaffine classifiers, one to predict the head of a given token and the other to predict

the resulting arc’s label. For input representation, the model uses BiLSTM modules to compute learned word embeddings and add them to their corresponding pretrained word embeddings that are later concatenated with corresponding part-of-speech (POS) embeddings. To ensure a well-formed tree at test time, the maximum spanning tree (MST) algorithm is used.

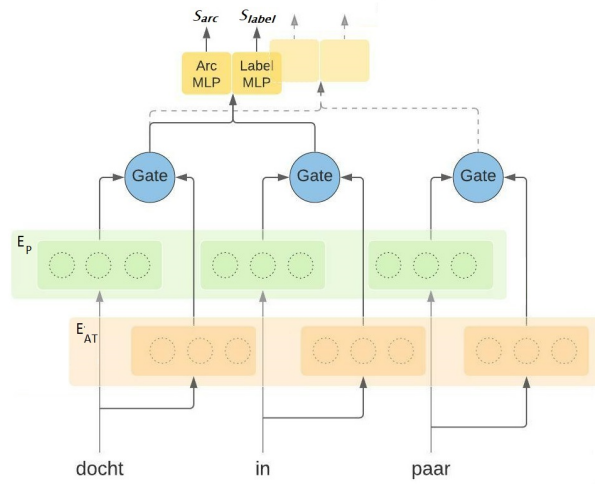


Figure 7.1. The parser architecture with semi-supervised auxiliary task enhancement.

E_P is the parser encoder, E_{AT} is the sequence labeler encoder trained on one of the auxiliary tasks. For a given token pair, the model calculates a weighted average of each token’s hidden representation from E_P and E_{AT} . The resulting vectors are given to two multi-layer perceptrons (MLP) to produce an arc score S_{arc} and a label score S_{label} for the given token pair. The input tokens are taken from the Frisian-Dutch Fame Treebank.

7.3.2. Semi-supervised Enhancement through Auxiliary Sequence Labeling Tasks

We follow Rotman and Reichart [157] to exploit unlabeled data for CS dependency parsing. Rather than directly using pseudo-labeled data as an additional source in training, the main idea is to extract and utilize parsing-related knowledge from automatically parsed data. This is achieved by training contextualized embedding models on a number of auxiliary sequence labeling tasks derived from the raw data parsed

by the base parser and then combining encoders of these trained models with that of the base parser through a gating procedure [166] as described in Section 7.3.3. Figure 7.1 depicts this enhanced parser. The combined model is then re-trained on the gold labeled data.

For their experimental setup, Rotman and Reichart [157] consider three token-level sequence labeling schemes to extract the structural information encoded in the parsed sentences. These are:

(i) *Number of Children (NOC)*. The task is to predict the number of children each token has in a dependency tree.

(ii) *Distance to the Root (DTR)*. Each token is tagged with its minimum distance to the root token of the dependency tree.

(iii) *Relative POS-based Encoding (RPE)*. Each token in a sentence is tagged with its head’s POS tag in a simplified form and its distance from the head. The distance calculation considers only the intermediate tokens that share the same POS tag with its head.

Although these three auxiliary tasks offer a comprehensive scheme in terms of extracting parsing-related knowledge from automatically parsed data, we search ways of channeling the embedded knowledge in parsed trees more thoroughly to the trained word embedding layers of the parser. We come up with three additional sequence labeling tasks:

(iv) *Language ID of Head (LIH)*. We start with CS-specific features of parsed trees. The most prominent of them is the language ID (LID) features of the tokens in CS treebanks. Considering the positive impact of LIDs in various other NLP tasks [35, 167, 168], we design a simple auxiliary sequence labeling task that makes use of LIDs. Unlike previous work using token LIDs, LIH tags each token with the LID of

its head. This way, information about the language of tokens with which each token tends to relate in terms of dependencies is conveyed to the learning model.

(v) *Simplified Morphology of Head (SMH)*. Morphological features are found to be beneficial in parsing morphologically-rich languages [169]. This was our motivation to create a new auxiliary task based on morphology. In the SMH scheme, each token is assigned its head’s morphological features. To reduce the number of labels, we use only a subset of the morphological features set (*Aspect*, *Case*, *Foreign*, *Mood*, *NumType*, *Person*, and *VerbForm* features), selected by considering the inclusiveness and the prevalence of the features across the data. The main idea of SMH is to provide morphological clues to the parser while also giving information about the structure of the tree.

A similar approach is also tried by Sandhan *et al.* [170]. They define a sequential task to predict the full set of morphological features for a given token. In our preliminary experiments, we observed that using the full set of morphological features does not improve the accuracy. In CS treebanks the unique number of features is increased due to the combination of language-specific feature sets of the language pair, making the task more complex. To reduce the complexity, we design SMH as utilizing only a subset of the morphological features of (not the token itself, but) the head of the token.

(vi) *Punctuation Count (PC)*. Lastly, we design the PC task that only needs root tokens unlike all other tasks that need parsed trees to function. PC is also not dependent on morphological, POS, or LID tags as SMH, RPE, and LIH tasks. PC tags each token with the number of punctuation between that token and the root token in the sentence. We observe a connection between the position of punctuation and phrase boundaries in a sentence which goes in line with previous studies [171,172]. PC roughly groups tokens into phrases that usually constitute sub-trees in a dependency tree.

Figure 7.2 shows the outputs of these tasks on the dependency tree of an example CS sentence.

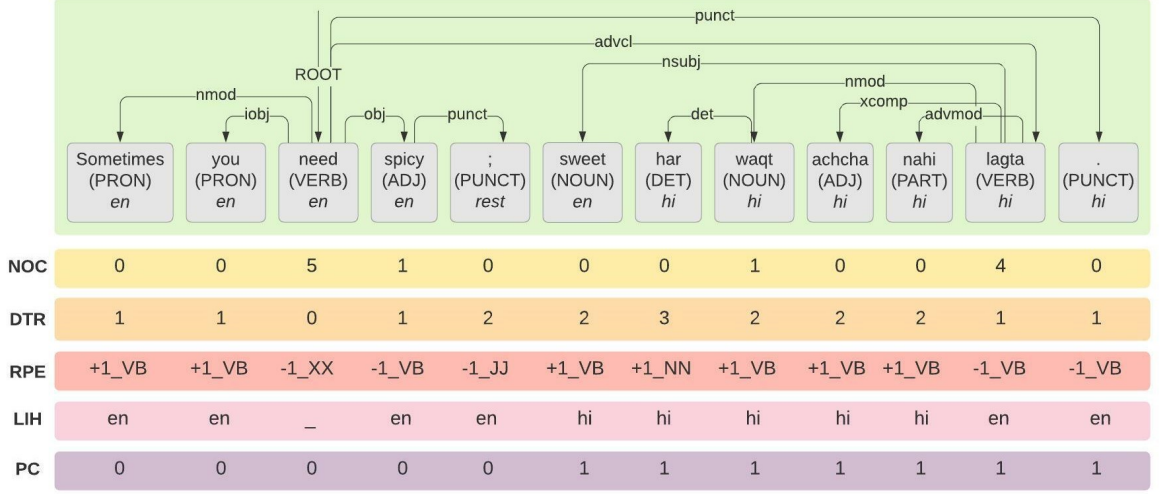


Figure 7.2. The dependency tree of an example sentence from Hindi-English HIENCS Treebank. Each node in the tree is tagged with the five auxiliary task schemes depicted in Section 7.3.2. Tags for the case of the SMH scheme are not shown for this example since the HIENCS Treebank does not include morphology.

7.3.3. The Gating Procedure

To create the final parser, the trained auxiliary task models are combined with the base parser through a gating mechanism [166] which learns to scale between the encoders of the auxiliary sequence labelers and that of the parser (see Fig. 7.1).

Formally, the combined representation can be formulated as:

$$b^t = \sigma(W^{gate}(e^{parser} \oplus e^{labeler}) + w^{gate})$$

$$g^t = b^t \cdot e^{parser} + (1 - b^t) \cdot e^{labeler},$$

where e^{parser} and $e^{labeler}$ are the outputs of the parser and sequence labeler encoders, respectively. \oplus denotes concatenation. W^{gate} and w^{gate} are the learned parameters of the gating procedure and σ is the sigmoid function. The final combined vector g^t is then given to the biaffine classifiers.

7.3.4. Transformer-based Adaptation of the Model

Our base parser as described in [117] has some shortcomings in the choice of the input representation, especially when the target language has very little or no training data and there is no accompanying pretrained word embeddings to represent the input. This is also the case with CS language pairs. In that situation, utilizing the expressive power of transformers can be a good solution. Pretrained on huge amounts of raw data in different languages, multilingual transformer-based language models have proven remarkably effective [44, 173, 174]. One such model is XLM-R [49]. Pretrained on text data in 100 languages, XLM-R shows SOTA performance in many languages including low-resource ones.

To the best of our knowledge, such a deep contextualized semi-supervised scheme has not been incorporated with XLM-R before. So, we re-implement the auxiliary task modules and the combined parsing approach for an XLM-R-based encoding module. For this purpose we follow the XLM-R-based parsing architecture of Grünewald *et al.* [175] which has the same biaffine parsing model described in [117]. Our aim is to observe how extracting parsing-related knowledge from semi-supervised auxiliary tasks affects a multilingual transformer model.

7.4. Experiments

7.4.1. Data

We perform experiments on all CS treebanks in Universal Dependencies (v2.8). These are Komi-Zyrian IKDP (Kpv-Ru), Hindi-English HIENCS (Hi-En), Frisian-Dutch Fame (Fy-Nl), and Turkish-German SAGT (Tr-De) treebanks. All except Hi-En are based on spoken CS data. Hi-En is constructed from bilingual tweets. Table 7.1 states basic statistics and related resources for each treebank.

Table 7.1. Some statistics and related resources for the CS treebanks. Fy-Nl is provided as a single test set of 400 utterances. As in [152], we split it into a development set (first 150 utterances) and a test set (remaining 250 utterances). CMI is the code-mixing index [176] showing how frequent code-switching happens in text.

	Kpv-Ru (Komi-Russian)	Hi-En (Hindi-English)	Fy-Nl (Frisian-Dutch)	Tr-De (Turkish-German)
Train	-	1,448	-	578
Dev	-	225	150	801
Test	214	225	250	805
CMI	16.97	36.08	17.80	28.78
Morphology	yes	no	no	yes
Monolingual treebanks	both	both	only Dutch	both
Unlabeled CS data	Komi Social Media	LinCE	FAME!	TuGeBiC
XLM-R	only Russian	both	both	both
FastText	only Russian	both	both	both

7.4.2. Training Setup

Due to lack of training data in some CS treebanks, we have two types of experimental setup. We train the parser models on *in-domain* data for Hi-En and Tr-De. In these experiments we use each treebank’s own training set. However, Kpv-Ru and Fy-Nl consist of a test set only. Hence, training of the latter two treebanks are on *out-of-domain* data. For Kpv-Ru which includes Komi-Russian code-switching, we train the models on Komi-Zyrian Lattice UD Treebank [160] of monolingual Komi data. The first 562 sentences in Komi-Zyrian Lattice are used for training, the remaining 100 are used for development. For Fy-Nl, our training data is the Dutch Alpino UD Treebank [63]. We chose Dutch Alpino over the other Dutch UD treebank (LassySmall) as Alpino is found more effective in parsing Fy-Nl [152].

7.4.2.1. Unlabeled Data.

Komi-Russian. Komi Social Media Corpus [177] is part of a social media corpora project for minority Uralic languages [178]. The data is crawled from *vkontakte*, a social

media service mostly popular in Russia. Collected texts are automatically separated to monolingual segments of *Komi*, *Russian*, or *Unknown* via a dictionary-based method. For our purposes, we extract 3,862 CS sentences from the corpus by joining consecutive segments that alternate between *Komi* and *Russian*.

Hindi-English. We employ the datasets in the LinCE CS benchmark [179] for this language pair. The benchmark provides three different corpora with gold LID and POS labels for Hindi-English [180–182]. We combine these three corpora to use them as unlabeled data. The resulting data consists of 10,989 sentences.

Frisian-Dutch. We extract CS sentences from the FAME! Corpus [183] which contains radio broadcasts in Frisian-Dutch. From this corpus, which is also the source of the Fy-Nl treebank, we select 2,170 sentences that include at least one CS point and are not already in the treebank.

Turkish-German. TuGeBiC⁷ is a set of transcriptions, collected from interviews with Turkish-German bilinguals in the 90s [184]. It contains 16,950 sentences. We use the whole corpus, and only remove the speaker IDs and metadata from the files.

7.4.2.2. Sequence Labeler Training. Training auxiliary models on sequence labeling tasks is done on automatically parsed version of the corresponding unlabeled data for each treebank. Some of the sequence labeling tasks need specific labels on unlabeled data to function. These are POS tags for RPE, LID labels for LIH, and morphological annotation for SMH. In training of these tasks, we use gold labels when available (POS tags for Hi-En; LIDs for Kpv-Ru, Hi-En, and Fy-Nl) and train taggers in the absence of gold labels (POS tags for Kpv-Ru, Fy-Nl, and Tr-De; LIDs for Tr-De; morphological features for Kpv-Ru and Tr-De).

⁷Available for research purposes. We obtained it by contacting Jeanine Treffers-Daller.

7.4.3. Baselines

As our baseline, we use Ma *et al.*' [185] re-implementation of the biaffine parser by Dozat *et al.* [117]. We call this model **Base_{LSTM}** as it uses BiLSTMs for contextualized word vectors.

As a second baseline, we implement the traditional self-training approach [154] in which the parser is first trained only on gold labeled data. Then, labels of unlabeled data are predicted by the trained parser. Finally the parser is re-trained on the combination of gold labeled data and pseudo-labeled data. We name this approach as **Self-training**.

For our experiments with XLM-R, we use Grünewald *et al.*' [175] implementation of the biaffine parser with XLM-R-based input representation. Input word embeddings are calculated as a weighted sum of all intermediate outputs of the transformer layers. Coefficients of the weighted sum are learned during the training phase. Apart from its multilingual transformer-based contextualized word representation model, it has the same biaffine parsing model in [117]. We call this version **Base_{XLMR}**.

Hyper-parameters of both parser models and sequence labelers are as follows:

Base_{LSTM}. We use Adam optimizer [146] with a learning rate of 0.002, batch size of 16, and all dropout probabilities are set to 0.33 for the parser and the sequence labeler models. We train the parser for 150 epochs and sequence labeling tasks for 100 epochs. We use 300-dimensional FastText embeddings [186] as pretrained word vectors. Since these embeddings are monolingual, we choose Russian FastText embeddings for Kpv-Ru, Hindi embeddings for Hi-En, Dutch embeddings for Fy-Nl, and Turkish embeddings for Tr-De treebanks. The model also uses 100-dimensional character embeddings and POS tag embeddings which are randomly initialized. The 3-layer BiLSTM modules of the parser and the sequence labeler have hidden layer size of 512 on each side. The decoder of the parser includes an arc MLP of size 512 and a label

MLP of size 128. The decoder of the sequence labeler consists of two fully connected layers with size 128 and 64, respectively.

Base_{XLMR}. Due to computational efficiency, we choose the 768-dimensional XLM-R base language model as the word representation module of the **Base_{XLMR}** architecture. For the parser, the arc MLP of the biaffine classifier has the same size with XLM-R model and the size of the label MLP is 256. Dropout for the classifier is set to 0.33. For the sequence labeler, we use a single-layer feed-forward neural network to extract logit vectors. We use AdamW optimizer [187] with a learning rate of 0.00004 and set batch size to 16. Number of epochs for the parser is 300 with an early stop of 50 epochs. For the sequence labeler, we train models for 100 epochs with an early stop of 15 epochs.

7.4.4. Semi-supervised Enhancement Models

We provide the list of enhancement models built on top of **Base_{LSTM}** and **Base_{XLMR}** where parser is combined with a sequence labeler trained on:

- **+NOC:** *Number of Children*,
- **+DTR:** *Distance to the Root*,
- **+RPE:** *Relative POS Encoding*,
- **+LIH:** *Language ID of Head*,
- **+SMH:** *Simplified Morphology of Head*,
- **+PC:** *Punctuation Count*.

Note that only Kpv-Ru and Tr-De treebanks have morphological annotation. Hence, **+SMH** is applied only to them. The **+PC** model is not applied to Fy-Nl since the treebank does not have punctuation.

Additionally, we perform experiments by ensembling more than one auxiliary task model with the base parser. We experiment with two configurations. First, we integrate *Number of Children*, *Distance to the Root*, and *Relative POS Encoding* models together (**+NOC, +DTR, +RPE**). This is also the ensemble configuration in Rotman

and Reichart [157]. Since we have additional three tasks, we also make the combination of three best performing models for each treebank and name this ensemble version as **+Best Combination**. Due to high memory consumption of XLM-R-based models, this ensemble technique cannot be applied to our XLM-R-based parsing architecture. For combining encoders of more than one auxiliary task model with the parser encoder, we use Rotman and Reichart’s [157] extension to the gating mechanism of Sato *et al.* [166]. We perform three runs for each model and report the average UAS and LAS scores.

7.5. Results

Table 7.2 shows the performance of all LSTM-based models and of the previous works on the test set of each treebank in terms of attachment scores. Significance testing is performed using the approximate randomization test [148] on the model outputs with the number of shuffles set to 5,000.

7.5.1. Parsing of Code-Switching Language Pairs

Comparison to Baselines. On all treebanks, the auxiliary task enhancement methods improve the scores when compared to **Base_{LSTM}** by 4.94 points in UAS and 3.86 points in LAS on average. The best performing enhancement model differs across treebanks. We observe the same pattern for the traditional self-training method. **Self-training** fails to surpass the proposed approach on any of the treebanks. Its parsing performance even falls below that of **Base_{LSTM}** on Kpv-Ru and Tr-De. It shows the highest improvement with respect to **Base_{LSTM}** on Fy-Nl. Yet, the best one of the auxiliary task enhancement methods significantly outperforms **Self-training** on each treebank.

New Individual Tasks. The **+LIH** model which employs LIDs performs best on Kpv-Ru, and second best on Hi-En. Its performance on Tr-De and Fy-Nl is comparable with the other models. It is also in the **Best Combination** ensemble for all treebanks. This indicates the importance of language IDs in CS dependency parsing.

Table 7.2. Attachment scores of baselines, our models, and the previous works on all CS UD treebanks. +SMH is not applicable to Hi-En and Fy-Nl due to the lack of morphology in these treebanks. +PC cannot be applied to Fy-Nl since it has no punctuation. †Best combination for each treebank: +NOC, +LIH, +PC for Kpv-Ru, +DTR, +RPE, +LIH for Hi-En, +NOC, +RPE, +LIH for Fy-Nl, and +RPE, +LIH, +SMH for Tr-De. The best scores for each dataset are underlined and bold. Scores marked with * significantly outperform both **Base_{LSTM}** and **Self-training**.

		Kpv-Ru		Hi-En		Fy-Nl		Tr-De	
		UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
<i>Baselines</i>	Base_{LSTM}	62.24	45.10	80.10	71.29	64.97	49.56	67.50	57.88
	Self-training	59.55	43.27	80.47	72.88	68.91	53.24	60.86	52.04
<i>Semi-supervised Enhancement</i>	+NOC	64.83*	46.53*	81.67	72.94	71.80*	53.35	70.86*	60.97*
	+DTR	64.80*	45.53	81.94	72.96	71.48*	53.10	70.88*	60.63*
	+RPE	64.95*	45.90	82.75*	73.84	72.98*	54.12	71.40*	61.46*
	+LIH	65.70*	47.13*	82.24*	73.54	72.20*	51.98	71.39*	61.46*
	+SMH	64.63*	45.31	-	-	-	-	71.41*	61.50*
	+PC	64.67*	46.79*	81.40	72.76	-	-	71.25*	61.44*
<i>Ensemble</i>	+NOC, +DTR, +RPE	65.59*	46.86*	82.75*	74.09*	73.97*	56.10*	70.55*	60.95*
	+Best Combination†	64.98*	46.22*	82.77*	74.02*	74.69*	56.39*	70.92*	61.65*
<i>Previous Work</i>	Bhat <i>et al.</i> [158]	-	-	80.23	71.03	-	-	-	-
	Braggaar and van der Goot [152]	-	-	-	-	70.20	55.60	-	-
	van der Goot <i>et al.</i> [153]	-	22.20	-	65.50	-	54.00	-	60.90
	Müller-Eberstein <i>et al.</i> [165]	-	-	73.62	62.66	-	-	66.75	55.04

The +SMH model which is only applied to Kpv-Ru and Tr-De is the best performing one on Tr-De. However, all other tasks outperform +SMH on Kpv-Ru. This might be due to the quality difference in morphological taggers trained on these treebanks. The morphological tagger we trained on the CS training set of Tr-De has an accuracy of 82% on its test set. However, to train a tagger for Kpv-Ru we used monolingual Komi data only. Accuracy of this tagger on Kpv-Ru test set is 66%. It seems the Kpv-Ru parser suffers from error propagation.

The simplest enhancement model +PC performs comparable to others, even outperforming +NOC and +DTR on Kpv-Ru and Tr-De. Since it only needs the root position in the sentence to perform, this model can be an alternative to other models when gold/predicted POS or morphological tags are hard to acquire. It can also be pre-

ferred when the error propagated to the auxiliary tasks from the base parser through predicted trees is high, damaging accuracy of the tasks that rely on these parses.

Individual Tasks vs Ensembles. Ensembling multiple tasks improves UAS and LAS on Hi-En and Fy-Nl and LAS on Tr-De when compared with the best performing single task. The **+Best Combination** ensemble works better on Fy-Nl and Tr-De than the **+NOC, +DTR, +RPE** ensemble proposed by Rotman and Reichart [157]. Looking at the overall results, we observe that including **+RPE** and **+LIH** together has a favorable effect on improving CS parsing performance.

Who Benefits Most and Least. Fy-Nl is the most benefited treebank from the proposed model. The best performing enhancement model **+Best Combination** on Fy-Nl achieves almost 10/7 points increase in UAS/LAS when compared with **Base_{LSTM}**. The least benefited treebank is Kpv-Ru with 2.5/1.1 points increase in UAS/LAS. Having similar amount of unlabeled data and no CS training data, these treebanks differ in their training data amounts. The Dutch Alpino Treebank used to train Fy-Nl models has 13,603 sentences whereas the Komi-Zyrian Lattice Treebank for Kpv-Ru models includes 662 sentences. So, automatic parsing of unlabeled data of Kpv-Ru by a model trained on 662 sentences can be much noisier than that of Fy-Nl. In Section 7.5.3, we discuss the effect of gold training data amount on the parsing performance.

Comparison to Previous Work. The best enhancement model always achieves better scores than previous state-of-the-art on each treebank. In this respect, the biggest improvement is observed on Kpv-Ru with more than 24 points increase in LAS. In addition, it should be noted that, model architectures are not quite comparable as some of the previous work use a lot more resources than our models. For instance, Müller-Eberstein *et al.* [165] perform data selection on whole UD datasets for training and utilize multilingual BERT.

Proposed Method and XLM-R. Attachment scores of **Base_{XLMR}** and our XLM-R adaptation of auxiliary task enhancement models are given in Table 7.3. Our first

observation is the limited performance of **Base_{XL_{MR}}** in parsing CS treebanks. We see that the enhancement models do not have the same impact on **Base_{XL_{MR}}** as they have on **Base_{L_{STM}}**. The only significant performance increase is on Fy-Nl where the best performing enhancement model **+NOC** outperforms **Base_{XL_{MR}}** by almost 2/1.5 points in UAS/LAS. For Kpv-Ru, only model that surpasses the baseline is **+RPE**. The difference is found statistically significant only in UAS. For Hi-En, all enhancement models except **+NOC** perform better than **Base_{XL_{MR}}**. Yet, the only significant improvement is achieved by **+DTR** in UAS. None of the enhancement models surpass **Base_{XL_{MR}}** on Tr-De but the difference between the scores is not found to be significant. Another remarkable observation is our models built on top of **Base_{L_{STM}}** outperforming all XLM-R-based models with the exception of Tr-De. This answers our second question: XLM-R is not always the best option. For powerful models like XLM-R, multilinguality can harm the performance when the target language is unknown to the model. Our results suggest that in such cases it is better to employ simpler models that are tailored for the exact task.

Table 7.3. Performance of XLM-R-based parser and our XLM-R adaptation of auxiliary task enhancement models. The best scores for each dataset are underlined and bold. Scores marked with * significantly outperform **Base_{XL_{MR}}**.

	Kpv-Ru		Hi-En		Fy-Nl		Tr-De	
	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS
Base_{XL_{MR}}	57.90	43.12	81.42	71.54	65.75	50.27	<u>75.93</u>	<u>66.30</u>
+NOC	57.09	42.79	81.28	71.58	<u>67.50</u>*	<u>51.64</u>*	75.79	65.98
+DTR	56.65	42.37	<u>82.15</u>*	71.89	66.85*	50.45	75.56	65.73
+RPE	<u>58.77</u>*	<u>43.84</u>	81.79	71.84	67.35*	51.13*	75.49	65.77
+LIH	57.24	43.19	81.92	<u>71.93</u>	66.26	50.10	75.51	65.78
+SMH	56.98	43.25	-	-	-	-	75.53	65.66
+PC	56.81	41.97	81.46	71.89	-	-	75.14	65.45

7.5.2. Comparison of Methods in terms of Computational Resources

Table 7.4 provides time and memory usage of $\text{Base}_{\text{LSTM}}$, $\text{Base}_{\text{XLMR}}$, and our proposed best model for each treebank. Labeled attachment scores (LAS) acquired by these models on each treebank are also given.

Table 7.4. Comparison of baselines and the proposed approach according to training time, memory usage during training, and LAS. Our best model on Kpv-Ru is the +LIH model. For all other treebanks, our best model is an ensemble that combines three task models.

	Kpv-Ru	Hi-En	Fy-Nl	Tr-De
<i>Training time</i>				
$\text{Base}_{\text{LSTM}}$	0h9m	0h15m	0h45m	0h20m
Our best model	0h25m	0h40m	2h30m	0h55m
$\text{Base}_{\text{XLMR}}$	3h40m	3h15m	11h0m	1h30m
<i>Memory usage (GB)</i>				
$\text{Base}_{\text{LSTM}}$	3.6	3.6	3.8	3.5
Our best model	4.5	7.6	7.3	7.4
$\text{Base}_{\text{XLMR}}$	9.9	7.9	9.6	8.4
<i>LAS</i>				
$\text{Base}_{\text{LSTM}}$	45.10	71.29	49.56	57.88
Our best model	47.13	74.09	56.39	61.65
$\text{Base}_{\text{XLMR}}$	43.12	71.54	50.27	66.30

From the table, we observe that there is a trade-off between performance and resource consumption for the three models. Training time of $\text{Base}_{\text{LSTM}}$ is the shortest. Yet, our best model improves the performance significantly at the expense of a slight increase in training time. $\text{Base}_{\text{XLMR}}$ has the longest training time by a large margin.

In terms of memory usage, there is a similar pattern to that of training time. $\text{Base}_{\text{LSTM}}$ needs approximately 50% less memory than our best model, yet there is on

average 3.86 points gap between LAS of the two models. $\text{Base}_{\text{XLRR}}$ is again the least preferable model here due to its highest memory consumption and low performance on parsing the treebanks with the exception of Tr-De. Only for Tr-De it outperforms the other two models and can be the model of choice for the parsing of Tr-De data.

7.5.3. Effect of Gold Labeled Data on the Parsing Performance

In our main experiments the gold training data size differs among the four datasets. While the gold labeled data used for training of Kpv-Ru and Tr-De includes approximately 500 sentences, Hi-En has 1,448 gold labeled training CS data and for Fy-Nl we used the training set of the Dutch Alpino UD Treebank which consists of 12,289 gold labeled Dutch sentences. In order to observe how the amount of gold labeled training data affects the models' performance, we did a set of experiments on each of Hi-En and Fy-Nl datasets by incrementally increasing the size of labeled training data from 500 to the original training data size as used in the main experiments. Figure 7.3 shows results of these experiments.

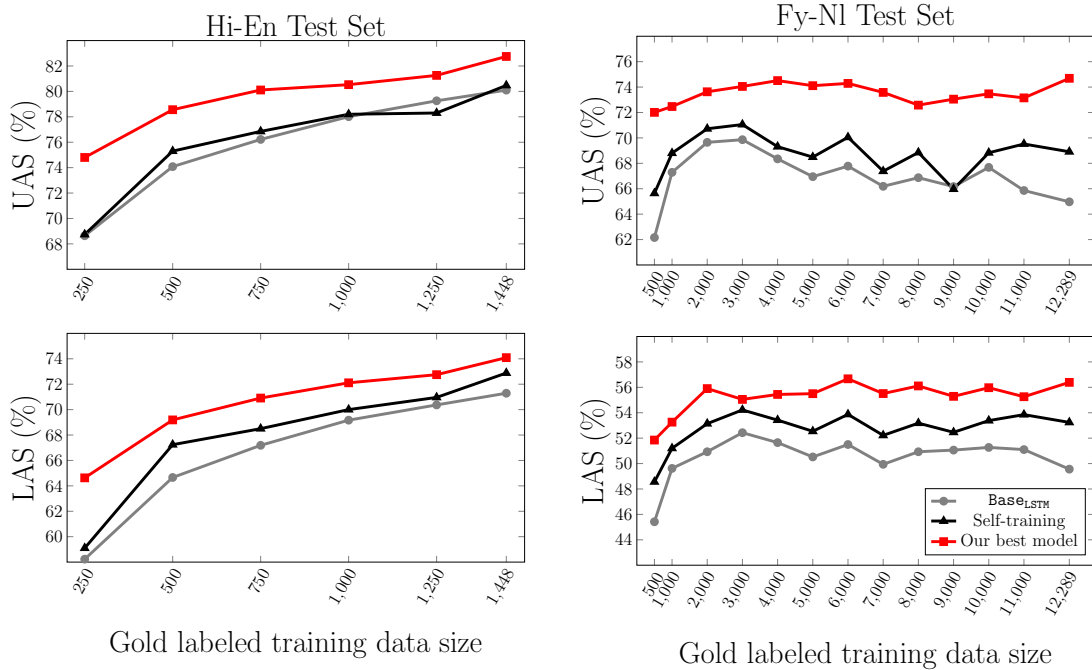


Figure 7.3. Comparison of $\text{Base}_{\text{LSTM}}$, Self-training, and our best model on Hi-En and Fy-Nl.

We observe that our best model on these datasets (+NOC, +DTR, +RPE for Hi-En and +NOC, +RPE, +LIH for Fy-Nl) always surpasses **Self-training** and **Base_{LSTM}** regardless of the available gold training data. Increasing the labeled training data has always a positive effect on the performance of all models for Hi-En but causes fluctuations in the performance for the case of Fy-Nl. The reason for this difference might be that the training data of Hi-En is in-domain and includes CS sentences, while the training data we use for Fy-Nl is out-of-domain and includes monolingual Dutch sentences.

Considering long training time and high resource consumption of the XLM-R-based parser (see Table 7.4) and success of our LSTM-based models, we suggest LSTM-based auxiliary task enhancement for low-resource dependency parsing of CS data.

7.5.4. Parsing of Turkish

We also applied the proposed models to the Turkish IMST Treebank [60], a mid-size monolingual UD treebank. Tables 7.5 and 7.6 show attachment scores of the baselines and the proposed methods on the LSTM-based and XLM-R-based architectures, respectively. In these experiments, we utilized the BOUN Treebank [11] as the unlabeled data for Turkish that is needed by semi-supervised models.

In Table 7.5, we observe that +SMH performs the best between the single task models and ensembling the +RPE, +SMH, and +PC tasks achieves the highest scores among the semi-supervised enhancement models. Similar to its performance on CS treebanks, the traditional self-training method failed to outperform the semi-supervised enhancement models in Turkish.

When we examine Table 7.6 for the performance of XLM-R-based models, we see an improvement on the scores by some of the semi-supervised enhancement models (+DTR and +SMH) over the **Base_{XLM-R}** but this difference is not found to be significant. However we observe that, in contrast to the experiment results on highly low-resource CS treebanks, XLM-R-based models outperform the LSTM-based ones by a large mar-

gin for the case of Turkish. This difference in the models' success might be due to the well-representation of Turkish in the pretrained XLM-R model. When comparing the two architectures, we see a trade-off between the remarkable success (7% increase in both UAS and LAS over the LSTM-based models) and higher resource consumption and training time of XLM-R-based models. Which side of this trade-off will be chosen should be decided by considering the existing resources, conditions, and needs.

Table 7.5. Attachment scores of LSTM-based baselines and our models on the test of the Turkish IMST Treebank. Scores marked with * significantly outperform both

Base_{LSTM} and **Self-training**.

		IMST	
		UAS	LAS
<i>Baselines</i>	Base_{LSTM}	68.76	62.33
	Self-training	67.51	61.70
<i>LSTM-based Semi-supervised Enhancement</i>	+NOC	69.61	62.64
	+DTR	70.02*	62.57
	+RPE	70.84*	63.97*
	+SMH	71.75*	65.06*
	+PC	70.63*	63.36*
	+NOC, +DTR, +RPE	71.68*	65.11*
	+RPE, +SMH, +PC	<u>72.57*</u>	<u>66.20*</u>

Table 7.6. Attachment scores of XLM-R-based baselines and our models on the test set of the Turkish IMST Treebank.

		IMST	
		UAS	LAS
<i>Baseline</i>	Base_{XLM-R}	79.63	73.47
<i>XLM-R-based Semi-supervised Enhancement</i>	+NOC	79.88	73.47
	+DTR	<u>79.98</u>	<u>73.71</u>
	+RPE	79.84	73.59
	+SMH	79.89	73.62
	+PC	79.66	73.47

7.6. Conclusion

In this chapter, we proposed a parsing model for parsing of low-resource languages. We addressed this problem with a focus on CS language pairs. We presented a semi-supervised auxiliary task enhancement to a graph-based neural parser and created novel sequence labeling tasks that are shown as useful in improving the parser’s success. Experimental results show that our enhancement technique achieves SOTA performance on all CS UD treebanks and helps better utilization of unlabeled data for CS dependency parsing. We combine our enhancement models with XLM-R to see their performance on a multilingual transformer-based model. Results demonstrate that the powerful XLM-R shows limited performance and fails to surpass our semi-supervised auxiliary task enhancement models. We also applied the proposed architecture to Turkish and reached the highest parsing scores on the Turkish IMST Treebank. Our implementation of the proposed sequence labeling tasks and the XLM-R-based enhancement technique as well as the trained models are publicly available for research purposes at [15].

8. TOOLS

In this chapter, the annotation tool mentioned in Chapter 4 and the hybrid parser tool mentioned in Chapter 6 are introduced.

8.1. BoAT Annotation Tool

This section is based on the Annotation Tool Section of our collaborative work published as [11].

Annotation tools are fundamental to the facilitation of the annotation process of many NLP tasks including dependency parsing. UD treebanks are re-annotated or annotated from scratch in line with the annotation guidelines of the UD framework [4]. There are several annotation tools that are showcased within the UD framework such as UD Annotatrix [188] and ConlluEditor [189]. These tools are mostly based on mouse-clicks, and provide graph view and/or text view. Morphological features are, in general, not easy to annotate/edit with the available tools. There are also annotation tools that have been developed for annotating Turkish treebanks [56, 74, 190, 191]. However, they are not specific to the UD framework. Apart from that, they do not have practical user interfaces regarding dependency parsing.

We present BoAT, a new annotation tool specifically designed for dependency parsing. To the best of our knowledge, it is the first tool that provides tree view and table view simultaneously. BoAT enables annotators to use both mouse clicks and keyboard shortcuts. In addition, unlike previous dependency parsing annotation tools which show morphological features as a whole, in BoAT, morphological features are parsed and expanded into multiple columns, as they are one of the most re-annotated fields according to the observations of the annotators of the BOUN Treebank. The enhanced presentation of morphological features is beneficial for annotators. Using BoAT, tokenization can be easily changed by splitting or joining tokens. This is a

useful property, especially for agglutinative languages since they have more suffixes, and tokenization may differ according to the used methods. The tool itself, however, is not specific to agglutinative languages and can be used for other languages as well.

BoAT is designed with the aim of presenting a user-friendly, compact, and practical manual annotation tool that is built upon the preferences of the annotators. It combines useful features from other tools such as changing the tokenization, using a validation mechanism, and taking notes with novel features such as combining tree and table views, parsing morphological features, and adding keyboard shortcuts to match the needs of the annotators for the dependency parsing task.

While developing BoAT, we received feedback from the treebank annotators in every step of the process. One crucial aspect of annotation is speed. Annotation tools are helpful in this regard but they are still open to advancement in terms of speed. The existing tools within the UD framework mostly rely on mouse clicks and dragging, and the usage of keyboard shortcuts is very limited. Unlike them, almost every possible action within BoAT can be carried out via both mouse clicks and keyboard shortcuts. We aim to decrease the time-wise and ergonomic load introduced by the use of a mouse and to increase speed accordingly.

We also added the note taking option being inspired by the BRAT [192], a web-based manual annotation tool for visualization and editing. While notes are specific to annotations in BRAT, they are specific to each sentence in our tool. This feature enabled the annotators to have better communication and have better reporting power.

8.1.1. Features

BoAT is a desktop annotation tool which is specifically designed for CoNNL-U files. It offers both tree view and table view as shown in Figure 8.1. The upper part of the screen shows the default table view while the lower part shows the tree view. Below we explain briefly the components and some of the properties of the tool.

Data Viewer

Prev [] Add Row [] Delete Row 0 Go Write your note here... Next

☒ ID ☒ FORM ☒ LEMMA ☒ UPOS ☒ XPOS ☒ FEATS ☒ HEAD ☒ DEPREL ☐ DEPS ☒ MISC
☐ Abbr ☐ Animacy ☐ Aspect ☒ Case ☐ Clusivity ☐ Definite ☐ Degree ☐ Echo ☐ Evident ☐ Foreign ☐ Gender ☐ Mood ☐ NounClass ☐ NumType
☒ Number ☐ Number[psor] ☐ Person ☐ Person[psor] ☐ Polarity ☐ Polite ☐ Poss ☐ PronType ☐ Reflex ☐ Register ☐ Tense ☐ VerbForm ☐ Voice

ID	FORM	LEMMA	UPOS	XPOS	FEATS	HEAD	DEPREL	MISC	Case	Number
+ 1	Biraları	bira	NOUN	Noun	Case=Acc Number=Plur Person=3	2	obj	—	Acc	Plur
+ 2	devirdikçe	devir	VERB	Verb	Aspect=Perf Mood=Ind Polarity=Pos Tense=Pres VerbForm=Conv	4	advcl	—	—	—
+ 3	merakım	merak	NOUN	Noun	Case=Nom Number=Sing Number[psor]=Sing Person=3 Person[psor]=1	4	nsubj	—	Nom	Sing
+ 4	azdı	az	VERB	Verb	Aspect=Perf Mood=Ind Number=Sing Person=3 Polarity=Pos Tense=Past	0	root	SpaceAfter=No	—	Sing
+ 5	.	.	PUNCT	Punc	—	4	punct	—	—	—

Biraları devirdikçe merakım azdı.

Figure 8.1. A screenshot from the BoAT tool. The sentence is taken from Example 7.

Tree view: The dependency tree of each sentence is visualized in the form of a graph. Instead of using flat view, hierarchical tree view is used. If the user hovers the mouse pointer over a token in the tree, the corresponding token in the sentence above the tree is highlighted which gives the user a linearly readable tree in order to increase readability and clarity. The tree view is based on the hierarchical view feature in the CoNNL-U Viewer offered by the UD framework.

Table view: Each sentence is shown along with its default fields which are ID, FORM, LEMMA, UPOS, XPOS, FEATS, HEAD, DEPREL, DEPS, and MISC. The morphological features denoted by the FEATS field are parsed into specific subfields. These subfields are a subset of universal and language-specific features in the UD framework. These subfields are optional in the table view; annotators can choose which subfields they want to see. They are stored in the CoNNL-U file in a concatenated manner.

Customizing table view:. Annotators can customize the table view according to their needs by using the checkboxes assigned to the fields and the subfields of the FEATS field shown above the parsed sentence. In this way, the user can organize the table view easily and obtain a clean view by removing the unnecessary fields when annotating. This customization ameliorates readability, and consequently the speed of the annotation. The example in Figure 8.1 shows a customized table view.

Actions in table view:. To ease the annotation process, the most frequently used functions are assigned to keyboard shortcuts. Moreover, annotators can jump to any sentence by simply typing the ID of the sentence. The value in a cell is edited by directly typing when the focus is on that cell. If one of the features is edited, the FEATS cell is updated accordingly.

Changing tokenization:. One of the biggest challenges in the annotation process is keeping track of the changes in the segment IDs when new segmentations are introduced. In BoAT, new tokens can be added or existing ones can be deleted to overcome tokenization problems generated during the pre-processing of the text. Moreover, annotating multiword expressions often comes at the cost of updating the segment IDs within a sentence in the case of misdetected multiword expressions due to faulty automatic tokenization. Annotators may need an easy way to split a word into two different units. We enabled the annotators to split or join words within our tool by clicking the cells in the first column of the table (written “+” or “-”) or using keyboard shortcuts, which permits a more accurate analysis of multiword expressions.

Validation:. Each tree is validated with respect to the field values before saving the sentence. If an error is detected in the annotated sentence, an error message is issued such as “unknown UPOS value”. The error is shown between the table view and the tree view.

Taking notes:. With the note feature, the annotator is able to take notes for each sentence as exemplified on the topmost line in Figure 8.1. Each note is attached

to the corresponding sentence and stored in a different file with the ID of the sentence.

8.1.2. Implementation

BoAT [12] is an open-source desktop application. The software is implemented in Python 3 along with PySide2 and regex modules. In addition, CoNNL-U viewer is utilized by adapting some part of the UD API library [193]. Resources consisting of a data folder, the tree view, and validate.py are adapted from the UD tools [194]. for validation check. The data folder is used without any changes while some modifications have been made to validate.py. BoAT is a cross-platform application since it runs on Linux, OS X, and Windows.

The BoAT tool was designed in accordance with the needs of the annotators, and it increases the speed and the consistency of the annotation process on the basis of the annotators' feedbacks. Currently, BoAT only supports the CoNNL-U format of UD since it was designed specifically for dependency parsing. In the future, it may be extended to support other formats such as the CoNNL-U Plus format [195].

8.2. BOUN-PARS

BOUN-Pars is an LSTM-based dependency parser developed for Turkish. It is based on Stanford's graph-based neural dependency parser [82] and uses linguistically oriented rules and benefits from morphological information of words [9].

BOUN-Pars creates dependency parse trees of Turkish sentences in CoNNL-U format. The pre-processing steps of parsing from raw text: the segmentation, morphological tagging, and lemmatization tasks are performed by a pretrained model by TurkuNLP pipeline [84].

The source code is written in Python language. BOUN-Pars is publicly available at [14]. It has also an online tool at [13]. The parsing performance of BOUN-Pars on

Turkish IMST Treebank is reported in Chapter 6 and in the original article [9].

8.2.1. Requirements

BOUN-Pars requires the following tools:

- Python 3.7
- Tensorflow 1.12
- Torch 0.4
- Keras
- requests
- h5py
- matplotlib
- flask
- numpy
- ufal.udpipe
- pyyaml
- configargparse

8.2.2. Usage of BOUN-Pars

To train new models, the following script can be executed.

```
python main.py
--save_dir saves/NAME-OF-THE-TRAINED-MODEL
--config_file config/parser-config.cfg
```

where `parser-config.cfg` is the configuration file which includes paths to data files and hyper-parameters of the model. An example configuration file can be reached at [14]. Note that `NAME-OF-THE-TRAINED-MODEL` should be replaced with a name for the model to be trained.

After training the new model, CoNNL-U files can be parsed with the following script:

```
python main.py
--save_dir saves/NAME-OF-THE-TRAINED-MODEL
parse TEST-DATA.conllu
--output_dir ./
--output_file TEST-DATA-PARSED.conllu
```

Users can also parse sample texts using the online demonstration tool of BOUN-Pars that is publicly available at [13].

8.2.3. Reproducing the Original Results

In order to reproduce the original results on the test set of the Turkish IMST Treebank, the following Python script should be executed using the test set file of the IMST Treebank.

```
import evaluate

with open("tr_imst-ud-test.conllu", "r") as f:
    conllu_text = f.read()
result = evaluate.parse_conllu(conllu_text)
print(result)
```

9. CONCLUSION

9.1. Summary of Contributions

In this thesis, we focused on dependency parsing of Turkish language and improved this task by proposing state-of-the-art methodologies. We approached the problem from two perspectives. First, we developed two novel parsing models: the hybrid parser that meets the specific needs of Turkish language and the semi-supervised parser that is especially designed considering low-resource languages. While developing these models we also manually annotated a new comprehensive Turkish dependency treebank as a solution to the inconsistent and insufficient annotated data problem of Turkish treebanking.

In Chapter 5, we performed our initial study [111] and investigated the effect of morphology to the parsing performance of an LSTM-based deep model. We developed Lemma-Suffix and Morphological Features embedding models for a transition-based parser [109] that employs character-based embeddings. We participated in CoNLL-18 shared task on multilingual dependency parsing [112] and evaluated these models on UD version 2.2 treebanks. From the results, we observed that incorporating morphological information to a character-based word embedding model achieves better parsing performance for most of the agglutinative languages in the evaluation set.

After seeing the promising effect of integrating morphology to the LSTM-based parsing model, we focused on creating a hybrid parser for Turkish by integrating linguistically-oriented rules and morphology into a deep dependency parsing approach. For this purpose, we first proposed a novel rule-based enhancement method that can be integrated to any neural dependency parser in Chapter 6.3.2. This method includes linguistically-oriented hand-crafted rules that are based on a comprehensive grammar book for Turkish [32]. We also proposed a morphology-based enhancement method with three different ways of including morphological information to the deep parser

in Chapter 6.3.4. In order to integrate these enhancement methods to the widely-used BiLSTM-based biaffine parser [82], we proposed a simple yet useful integration method that allows to combine the proposed enhancement methods with any neural dependency parser via its dense representation. The final hybrid parser acquired state-of-the-art performance on parsing of Turkish IMST Treebank, even outperforming a famous multilingual BERT-based parser, Udify [147].

In Chapter 7, we expanded the focus of this thesis by proposing a novel parsing method with semi-supervised enhancement for low-resource languages. In this work, we followed a deep contextualized self-training approach [157] and created semi-supervised auxiliary tasks to enhance the BiLSTM-based biaffine parser [117]. We trained parsing-related auxiliary sequence labeling tasks on pseudo-labeled data and combined these trained auxiliary task models with the base parser through a gating mechanism [166]. We evaluated this parser on both Turkish and low-resource code-switching language pairs and reached state-of-the-art performance on all treebanks. We observed that the new sequence labeling tasks that we introduced are found to be useful in improving the parsing performance. We also adapted the proposed methodology to the powerful XLM-R model [49] and discussed its effectiveness when combined with XLM-R-based word representation for dependency parsing.

In the meantime, we created resources and tools to be used in Turkish dependency parsing. In Chapter 4, we introduced our new BOUN Treebank consisting of 9,761 manually annotated sentences in Turkish. While creating the treebank, we ensured that the annotations are of high quality and consistent by considering linguistic properties of the language and by staying faithful to the UD annotation scheme. We collaborated with linguists for this task and performed a cross-checking process to assure consistency between annotations. Besides annotating syntactic relations, we encoded universal POS tags, lemmas, and morphological features for each token in the treebank. We explained our linguistic decisions and annotation scheme in detail by providing examples for the challenging issues that are present in the BOUN Treebank.

To ease and accelerate the annotation process, we built an annotation tool, BoAT, specifically designed for dependency parsing. The BoAT tool is tailored to the needs of annotators and it increases the speed and consistency of the annotation process. We explained the features and usage of the BoAT tool in Chapter 8.1.

Creating the end-products and making them publicly available is as important a step as developing the underlying theoretical models. In Chapter 8.2, we introduced BOUN-Pars, an online and downloadable parsing tool based on the hybrid-parser presented in Chapter 6. We believe the products of this thesis that are made available to the public will contribute to Turkish dependency parsing, as well as to various fields of natural language processing.

In this thesis, we focused on dependency parsing of Turkish and presented two approaches for this purpose. In our first approach we proposed a rule-based and three morphology-based enhancement techniques for better representation of words in Turkish dependency parsing. Our experimental results on Turkish IMST Treebank showed the positive effect of combining simple linguistic rules and morphology with the deep architecture of a parsing model to the automatic parsing of Turkish. Our second approach to dependency parsing is a semi-supervised deep parsing model which utilizes auxiliary tasks trained on pseudo-labeled data to enhance the parsing performance. We created novel sequential labeling tasks to be used in training of the two different parsing models. Our approach acquired state-of-the-art scores on both Turkish and low-resource code-switching language pairs. According to the results of performed experiments, the proposed techniques are helpful for dependency parsing in the presence of data scarcity. Furthermore, we contributed a new comprehensive Turkish dependency treebank to this field. With detailed experiments, we showed that deep learning-based parsers that are extremely data-demanding always benefit from additional training data when this new data is high-quality and consistent with the original data.

9.2. Future Work

The hybrid dependency parsing approach presented in Chapter 6 was designed for Turkish. Our future directions for this study will be focused on adapting the proposed rule-based and morphology-based enhancement techniques to other suitable languages with restricted amount of annotated data. Rules’ individual accuracies can also be improved to increase the overall performance. Finding new methodologies to reduce the false-positive rates for each rule can be a good start for boosting their performance.

Semi-supervised deep dependency parser introduced in Chapter 7 showed promising success in dependency parsing of code-switched texts. As future work, we plan to extend this study by applying the proposed methodology to treebanks of all low-resource languages in the Universal Dependencies project. These languages might benefit from new auxiliary tasks tailored to demand their specific needs as in the case for the code-switching language pairs that highly gained from the Language ID of Head (LIH) task.

REFERENCES

1. Bārzdīņš, G., N. Grūzītis, G. Nešpore and B. Saulīte, “Dependency-Based Hybrid Model of Syntactic Analysis for the Languages with a Rather Free Word Order”, *Proceedings of the 16th Nordic Conference of Computational Linguistics*, pp. 13–20, Tartu, Estonia, 2007.
2. Sulubacak, U., G. Eryiğit and T. Pamay, “IMST: A Revisited Turkish Dependency Treebank”, *Proceedings of the 1st International Conference on Turkic Computational Linguistics (TurCLing)*, pp. 1–6, Konya, Turkey, 2016.
3. Zeman, D., “Universal Dependencies Version 2.9”, 2021, <http://hdl.handle.net/11234/1-4611>, accessed in April 2022.
4. Nivre, J., M.-C. de Marneffe, F. Ginter, Y. Goldberg, J. Hajič, C. D. Manning, R. McDonald, S. Petrov, S. Pyysalo, N. Silveira, R. Tsarfaty and D. Zeman, “Universal Dependencies v1: A Multilingual Treebank Collection”, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC)*, pp. 1659–1666, Portorož, Slovenia, 2016.
5. Mikolov, T., K. Chen, G. Corrado and J. Dean, “Efficient Estimation of Word Representations in Vector Space”, *Proceedings of the 1st International Conference on Learning Representations (ICLR) Workshop Track*, Arizona, USA, 2013.
6. Hall, J., J. Nilsson, J. Nivre, G. Eryiğit, B. Megyesi, M. Nilsson and M. Saers, “Single Malt or Blended? A Study in Multilingual Parser Optimization”, *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pp. 933–939, Prague, Czech Republic, 2007.
7. Eryiğit, G., “The Impact of Automatic Morphological Analysis & Disambiguation

- on Dependency Parsing of Turkish”, *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC)*, pp. 1960–1965, İstanbul, Turkey, 2012.
8. Çetinoğlu, Ö., “Turkish Treebank as a Gold Standard for Morphological Disambiguation and Its Influence on Parsing”, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC)*, pp. 3360–3365, Reykjavik, Iceland, 2014.
 9. Özateş, Ş. B., A. Özgür, T. Güngör and B. Öztürk, “A Hybrid Approach to Dependency Parsing: Combining Rules and Morphology with Deep Learning”, arXiv:2002.10116 [cs], 2020.
 10. Özateş, Ş. B., A. Özgür, T. Güngör and Ö. Çetinoğlu, “Improving Code-Switching Dependency Parsing with Semi-Supervised Auxiliary Tasks”, *Findings of the Association for Computational Linguistics: NAACL-HLT*, 2022.
 11. Türk, U., F. Atmaca, Ş. B. Özateş, G. Berk, S. T. Bedir, A. Köksal, B. Ö. Başaran, T. Güngör and A. Özgür, “Resources for Turkish Dependency Parsing: Introducing the BOUN Treebank and the BoAT Annotation Tool”, *Language Resources and Evaluation*, Vol. 56, No. 1, pp. 259–307, 2022.
 12. Köksal, A. and G. Berk, “BoAT: BOUN Annotation Tool for Dependency Parsing”, 2019, <https://github.com/boun-tabi/BoAT>, accessed in January 2020.
 13. Özateş, Ş. B., “BOUN-Pars Online Tool”, 2020, <https://tabilab.cmpe.boun.edu.tr/boun-pars>, accessed in April 2022.
 14. Özateş, Ş. B., “BOUN-Pars Source Code”, 2020, <https://github.com/sb-b/BOUN-PARS>, accessed in April 2022.
 15. Özateş, Ş. B., “Semi-supervised CS Dependency Parser Source Code”, 2022, <https://github.com/sb-b/ss-cs-depparser>, accessed in April 2022.

16. Bickel, B. and J. Nichols, *Inflectional Synthesis of the Verb*, Max Planck Institute for Evolutionary Anthropology, Leipzig, 2013.
17. Kapan, A., *Derivational Networks of Nouns and Adjectives in Turkish*, M.S. Thesis, Boğaziçi University, 2019.
18. Sak, H., T. Güngör and M. Saraçlar, “Turkish Language Resources: Morphological Parser, Morphological Disambiguator and Web Corpus”, *Proceedings of the International Conference on Natural Language Processing (ICLR)*, pp. 417–427, Gothenburg, Sweden, 2008.
19. Erguvanlı Taylan, E., “Pronominal versus Zero Representation of Anaphora in Turkish”, *Studies in Turkish Linguistics*, Vol. 209, p. 233, 1986.
20. Hoffman, B., *The Computational Analysis of the Syntax and Interpretation of “Free” Word Order in Turkish*, Ph.D. Dissertation, University of Pennsylvania, 1995.
21. Kural, M., “Postverbal Constituents in Turkish and the Linear Correspondence Axiom”, *Linguistic Inquiry*, Vol. 28, No. 3, pp. 498–519, 1997.
22. İşsever, S., “Information Structure in Turkish: The Word Order-Prosody Interface”, *Lingua*, Vol. 113, No. 11, pp. 1025–1053, 2003.
23. Kornfilt, J., *Asymmetries between Pre-Verbal and Post-Verbal Scrambling in Turkish*, pp. 163–180, De Gruyter Mouton, 2008.
24. Öztürk, B., “Non-configurationality: Free Word Order and Argument Drop in Turkish”, *The Limits of Syntactic Variation*, pp. 411–440, 2008.
25. Öztürk, B., *Postverbal Constituents in SOV Languages*, pp. 270–305, Oxford University Press Oxford, 2013.

26. Özsoy, A. S., *Word Order in Turkish*, Springer, Berlin, 2019.
27. Slobin, D. I. and T. G. Bever, “Children Use Canonical Sentence Schemas: A Crosslinguistic Study of Word Order and Inflections”, *Cognition*, Vol. 12, No. 3, pp. 229–265, 1982.
28. Erguvanlı Taylan, E., *The Phonology and Morphology of Turkish*, Boğaziçi University, İstanbul, 2015.
29. Kornfilt, J., *Case Marking, Agreement, and Empty Categories in Turkish*, Harvard University, Cambridge, 1984.
30. Özsoy, A. S., “Null Subject Parameter and Turkish”, *Studies on Modern Turkish: Proceedings of the Third Conference on Turkish Linguistics*, pp. 82–90, Tilburg, 1988.
31. Öztürk, B., *Null Arguments and Case-driven Agree in Turkish*, pp. 268–287, John Benjamins Publishing Company, 2006.
32. Göksel, A. and C. Kerslake, *Turkish: A Comprehensive Grammar*, Comprehensive Grammars, Routledge, Oxfordshire, 2005.
33. Auer, P. and L. Wei, *Handbook of Multilingualism and Multilingual Communication*, De Gruyter Mouton, Berlin, 2007.
34. Çetinoğlu, Ö., S. Schulz and N. T. Vu, “Challenges of Computational Processing of Code-Switching”, *Proceedings of the Second Workshop on Computational Approaches to Code Switching*, pp. 1–11, Austin, Texas, 2016.
35. Özateş, Ş. B. and Ö. Çetinoğlu, “A Language-aware Approach to Code-switched Morphological Tagging”, *Proceedings of the Fifth Workshop on Computational Approaches to Linguistic Code-Switching*, pp. 72–83, Online, 2021.

36. Çetinoğlu, Ö. and Ç. Çöltekin, “Two Languages, One Treebank: Building a Turkish–German Code-Switching Treebank and Its Challenges”, *Language Resources and Evaluation*, pp. 1–35, 2022.
37. Nivre, J., “Dependency Grammar and Dependency Parsing”, *MSI Report*, Vol. 5133, No. 1959, pp. 1–32, 2005.
38. Nivre, J. and R. McDonald, “Integrating Graph-Based and Transition-Based Dependency Parsers”, *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-HLT)*, pp. 950–958, Columbus, USA, 2008.
39. McDonald, R., F. Pereira, K. Ribarov and J. Hajič, “Non-Projective Dependency Parsing using Spanning Tree Algorithms”, *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT-EMNLP)*, pp. 523–530, Vancouver, Canada, 2005.
40. Dyer, C., M. Ballesteros, W. Ling, A. Matthews and N. A. Smith, “Transition-Based Dependency Parsing with Stack Long Short-Term Memory”, *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the Seventh International Joint Conference on Natural Language Processing (ACL-IJCNLP)*, Beijing, China, 2015.
41. Kiperwasser, E. and Y. Goldberg, “Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations”, *Transactions of the Association for Computational Linguistics*, Vol. 4, pp. 313–327, 2016.
42. Wang, W. and B. Chang, “Graph-Based Dependency Parsing with Bidirectional LSTM”, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 2306–2315, Berlin, Germany, 2016.
43. Sak, H., A. W. Senior and F. Beaufays, “Long Short-Term Memory Based Re-

- current Neural Network Architectures for Large Vocabulary Speech Recognition”, *Proceedings of the 15th Annual Conference of the International Speech Communication Association*, pp. 338–342, Singapore, 2014.
44. Devlin, J., M.-W. Chang, K. Lee and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pp. 4171–4186, Minneapolis, USA, 2019.
 45. Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, “Attention is All you Need”, *Advances in Neural Information Processing Systems*, Vol. 30, pp. 5998–6008, 2017.
 46. Wu, Y., M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, “Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation”, arXiv:1609.08144 [cs], 2016.
 47. Kudo, T. and J. Richardson, “SentencePiece: A Simple and Language Independent Subword Tokenizer and Detokenizer for Neural Text Processing”, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP): System Demonstrations*, pp. 66–71, Brussels, Belgium, 2018.
 48. Wenzek, G., M.-A. Lachaux, A. Conneau, V. Chaudhary, F. Guzmán, A. Joulin and E. Grave, “CCNet: Extracting High Quality Monolingual Datasets from Web Crawl Data”, *Proceedings of the 12th Language Resources and Evaluation Conference (LREC)*, pp. 4003–4012, Marseille, France, 2020.
 49. Conneau, A., K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer and V. Stoyanov, “Unsupervised Cross-lingual Representation Learning at Scale”, *Proceedings of the 58th Annual Meeting of the*

- Association for Computational Linguistics (ACL)*, pp. 8440–8451, Online, 2020.
50. Zeman, D. and J. Hajič, “CoNLL 2018 Shared Task Evaluation”, 2018, <https://universaldependencies.org/conll18/evaluation.html>, accessed in June 2018.
 51. Marcus, M. P., B. Santorini and M. A. Marcinkiewicz, “Building a Large Annotated Corpus of English: The Penn Treebank”, *Computational Linguistics*, Vol. 19, No. 2, pp. 313–330, 1993.
 52. Leech, G. and R. Garside, “Running a Grammar Factory: The Production of Syntactically Analysed Corpora or Treebanks”, *English Computer Corpora: Selected Papers and Research Guide*, pp. 15–32, 1991.
 53. Sampson, G., *English for the Computer: The SUSANNE Corpus and Analytic Scheme*, Oxford University Press, Oxford, 1995.
 54. Foth, K. A., A. Köhn, N. Beuck and W. Menzel, “Because Size Does Matter: The Hamburg Dependency Treebank”, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC)*, pp. 2326–2333, Reykjavik, Iceland, 2014.
 55. Borges Völker, E., M. Wendt, F. Hennig and A. Köhn, “HDT-UD: A Very Large Universal Dependencies Treebank for German”, *Proceedings of the Third Workshop on Universal Dependencies (UDW, SyntaxFest)*, pp. 46–57, Paris, France, 2019.
 56. Atalay, N. B., K. Oflazer and B. Say, “The Annotation Process in the Turkish Treebank”, *Proceedings of Fourth International Workshop on Linguistically Interpreted Corpora (LINC)*, Budapest, Hungary, 2003.
 57. Oflazer, K., B. Say, D. Z. Hakkani-Tür and G. Tür, *Building a Turkish Treebank*, pp. 261–277, Springer Netherlands, Dordrecht, 2003.

58. Megyesi, B., B. Dahlqvist, É. Á. Csató and J. Nivre, “The English-Swedish-Turkish Parallel Treebank”, *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC)*, pp. 17–23, Valletta, Malta, 2010.
59. Sulger, S., M. Butt, T. H. King, P. Meurer, T. Laczkó, G. Rákosi, C. B. Dione, H. Dyvik, V. Rosén, K. De Smedt, A. Patejuk, Ö. Çetinoğlu, I. W. Arka and M. Mistica, “ParGramBank: The ParGram Parallel Treebank”, *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 550–560, Sofia, Bulgaria, 2013.
60. Sulubacak, U., M. Gökırmak, F. Tyers, Ç. Çöltekin, J. Nivre and G. Eryiğit, “Universal Dependencies for Turkish”, *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pp. 3444–3454, Osaka, Japan, 2016.
61. Aksan, Y., M. Aksan, A. Koltuksuz, T. Sezer, Ü. Mersinli, U. U. Demirhan, H. Yilmazer, G. Atasoy, S. Öz, İ. Yıldız and Ö. Kurtoğlu, “Construction of the Turkish National Corpus (TNC)”, *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC)*, pp. 3223–3227, İstanbul, Turkey, 2012.
62. Brants, S., S. Dipper, S. Hansen, W. Lezius and G. Smith, “The TIGER Treebank”, *Proceedings of the Workshop on Treebanks and Linguistic Theories*, pp. 24–41, Sozopol, Bulgaria, 2002.
63. van der Beek, L., G. Bouma, R. Malouf and G. van Noord, “The Alpino Dependency Treebank”, *Proceedings of the Computational Linguistics in the Netherlands*, pp. 8–22, Twente, The Netherlands, 2002.
64. Say, B., D. Zeyrek, K. Oflazer and U. Özge, “Development of a Corpus and a Treebank for Present-day Written Turkish”, *Proceedings of the 11th International Conference of Turkish Linguistics*, pp. 183–192, Famagusta, North Cyprus, 2002.

65. Eryiğit, G. and T. Pamay, “ITU Validation Set for Metu-Sabancı Turkish Treebank”, *Türkiye Bilişim Vakfı Bilgisayar Bilimleri ve Mühendisliği Dergisi*, Vol. 7, No. 1, pp. 31–37, 2007.
66. Megyesi, B., B. Dahlqvist, E. Pettersson and J. Nivre, “Swedish-Turkish Parallel Treebank”, *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC)*, Marrakech, Morocco, 2008.
67. Brants, T., “TnT – A Statistical Part-of-Speech Tagger”, *Proceedings of the Sixth Applied Natural Language Processing Conference*, pp. 224–231, Seattle, Washington, USA, 2000.
68. Megyesi, B., *Data-Driven Syntactic Analysis - Methods and Applications for Swedish*, Ph.D. Dissertation, KTH Royal Institute of Technology, 2002.
69. Oflazer, K., “Two-Level Description of Turkish Morphology”, *Literary and Linguistic Computing*, Vol. 9, No. 2, pp. 137–148, 1994.
70. Yüret, D. and F. Türe, “Learning Morphological Disambiguation Rules for Turkish”, *Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL)*, pp. 328–334, New York, USA, 2006.
71. Nivre, J., J. Hall, J. Nilsson, A. Chanev, G. Eryiğit, S. Kübler, S. Marinov and E. Marsi, “MaltParser: A Language-Independent System for Data-Driven Dependency Parsing”, *Natural Language Engineering*, Vol. 13, No. 2, pp. 95–135, 2007.
72. Nivre, J., J. Nilsson and J. Hall, “Talbanken05: A Swedish Treebank with Phrase Structure and Dependency Annotation”, *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC)*, Genoa, Italy, 2006.
73. Zeman, D., M. Popel, M. Straka, J. Hajic, J. Nivre, F. Ginter, J. Luotolahti,

- S. Pyysalo and S. Petrov, “CoNLL Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies”, *Proceedings of the CoNLL Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pp. 1–19, Vancouver, Canada, 2017.
74. Pamay, T., U. Sulubacak, D. Torunoğlu-Selamet and G. Eryiğit, “The Annotation Process of the ITU Web Treebank”, *Proceedings of The Ninth Linguistic Annotation Workshop*, pp. 95–101, Denver, Colorado, USA, 2015.
 75. Sulubacak, U. and G. Eryiğit, “Implementing Universal Dependency, Morphology, and Multiword Expression Annotation Standards for Turkish Language Processing”, *Turkish Journal of Electrical Engineering & Computer Sciences*, Vol. 26, No. 3, pp. 1662–1672, 2018.
 76. Kayadelen, T., A. Öztürel and B. Bohnet, “A Gold Standard Dependency Treebank for Turkish”, *Proceedings of The 12th Language Resources and Evaluation Conference (LREC)*, pp. 5156–5163, Marseille, France, 2020.
 77. Çetinoğlu, Ö. and Ç. Çöltekin, “Challenges of Annotating a Code-Switching Treebank”, *Proceedings of the 18th International Workshop on Treebanks and Linguistic Theories (TLT, SyntaxFest)*, pp. 82–90, Paris, France, 2019.
 78. Çöltekin, Ç., “A Grammar-Book Treebank of Turkish”, *Proceedings of the 14th Workshop on Treebanks and Linguistic Theories (TLT)*, pp. 35–49, Warsaw, Poland, 2015.
 79. Çöltekin, Ç., “A Freely Available Morphological Analyzer for Turkish”, *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC)*, Valletta, Malta, 2010.
 80. Türk, U., F. Atmaca, Ş. B. Özateş, B. Öztürk Başaran, T. Güngör and A. Özgür, “Improving the Annotations in the Turkish Universal Dependency Treebank”,

- Proceedings of the Third Workshop on Universal Dependencies (UDW, SyntaxFest)*, pp. 108–115, Paris, France, 2019.
81. Türk, U., F. Atmaca, Ş. B. Özateş, A. Köksal, B. Öztürk Başaran, T. Güngör and A. Özgür, “Turkish Treebanking: Unifying and Constructing Efforts”, *Proceedings of the 13th Linguistic Annotation Workshop*, pp. 166–177, Florence, Italy, 2019.
 82. Dozat, T., P. Qi and C. D. Manning, “Stanford’s Graph-Based Neural Dependency Parser at the CoNLL 2017 Shared Task”, *Proceedings of the CoNLL Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pp. 20–30, Vancouver, Canada, 2017.
 83. Türk, U., F. Atmaca, Ş. B. Özateş, G. Berk, S. T. Bedir, A. Köksal, B. Ö. Başaran, T. Güngör and A. Özgür, “UD Turkish BOUN Treebank”, 2020, https://github.com/UniversalDependencies/UD_Turkish-BOUN, accessed in January 2020.
 84. Kanerva, J., F. Ginter, N. Miekka, A. Leino and T. Salakoski, “Turku Neural Parser Pipeline: An End-to-End System for the CoNLL 2018 Shared Task”, *Proceedings of the CoNLL Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pp. 133–142, Brussels, Belgium, 2018.
 85. Sak, H., T. Güngör and M. Saraçlar, “Resources for Turkish Morphological Processing”, *Language Resources and Evaluation*, Vol. 45, No. 2, pp. 249–261, 2011.
 86. Çöltekin, Ç., “(When) do We Need Inflectional Groups?”, *Proceedings of The First International Conference on Turkic Computational Linguistics*, pp. 38–43, Konya, Turkey, 2016.
 87. Tyers, F. M., J. Washington, Ç. Çöltekin and A. Makazhanov, “An Assessment of Universal Dependency Annotation Guidelines for Turkic Languages”, *Proceedings*

- of the Fifth International Conference on Turkic Languages Processing (TurkLang)*, p. 276, Kazan, Tatarstan, 2017.
88. TABILAB, “Universal Dependencies Online Documentation Edited by TABILAB”, 2020, https://github.com/boun-tabi/UD_docs, accessed in April 2022.
 89. Çetinoglu, Ö., *A Large Scale LFG Grammar for Turkish*, Ph.D. Dissertation, Sabancı University, 2009.
 90. Erguvanlı Taylan, E., *The Function of Word Order in Turkish Grammar*, University of California Press, Oakland, 1984.
 91. Kural, M., *Properties of Scrambling in Turkish*, M.S. Thesis, University of California, Los Angeles, 1992.
 92. Aygen, G., “Extractability and the Nominative Case Feature on Tense”, *Proceedings of the Tenth International Conference in Turkish Linguistics*, İstanbul, Turkey, 2003.
 93. İşsever, S., “Towards a Unified Account of Clause-Initial Scrambling in Turkish: A Feature Analysis”, *Turkic Languages*, Vol. 11, No. 1, pp. 93–123, 2007.
 94. Dependencies, U., “Universal Dependency Relations”, 2014, <https://universaldependencies.org/u/dep/index.html>, accessed in January 2020.
 95. Nivre, J., D. Zeman, F. Ginter and F. M. Tyers, “EACL 2017 Tutorial on Universal Dependencies”, 2017, <http://universaldependencies.org/eacl17tutorial/>, accessed in April 2019.
 96. Göksel, A., *The Auxiliary Ol at the Morphology-Syntax Interface*, John Benjamins, Amsterdam, 2001.

97. Hayashi, T., “The Dual Status of Possessive Compounds in Modern Turkish”, *Symbolae Turcologicae*, Vol. 6, pp. 119–129, 1996.
98. Kunduracı, A., *Turkish Noun-Noun Compounds: A Process-Based Paradigmatic Account*, Ph.D. Dissertation, University of Calgary, 2013.
99. Erguvanlı Taylan, E. and B. Öztürk Başaran, “The Notorious -(s)I(n) in Turkish: Neither an Agreement nor a Compound Marker?”, *Dilbilim Araştırmaları Dergisi*, Vol. 2, pp. 181–199, 2014.
100. Öztürk, B. and E. Erguvanlı Taylan, “Possessive Constructions in Turkish”, *Lingua*, Vol. 182, pp. 88–108, 2016.
101. Sağ, Y., *The Semantics of Number Marking: Reference to Kinds, Counting, and Optional Classifiers*, Ph.D. Dissertation, Rutgers University, 2019.
102. Zeman, D., “Core Arguments in Universal Dependencies”, *Proceedings of the Fourth International Conference on Dependency Linguistics*, pp. 287–296, Pisa, Italy, 2017.
103. Przepiórkowski, A. and A. Patejuk, “Arguments and Adjuncts in Universal Dependencies”, *Proceedings of the 27th International Conference on Computational Linguistics*, pp. 3837–3852, Santa Fe, New Mexico, USA, 2018.
104. Makazhanov, A., A. Sultangazina, O. Makhambetov and Z. Yessenbayev, “Syntactic Annotation of Kazakh: Following the Universal Dependencies Guidelines. A report”, *Proceedings of the Third International Conference on Turkic Languages Processing, (TurkLang)*, pp. 338–350, Kazan, Tatarstan, 2015.
105. Eryiğit, G., J. Nivre and K. Oflazer, “Dependency Parsing of Turkish”, *Computational Linguistics*, Vol. 34, No. 3, pp. 357–389, 2008.
106. Durgar El-Kahlout, İ., A. A. Akın and E. Yılmaz, “Initial Explorations in Two-

- phase Turkish Dependency Parsing by Incorporating Constituents”, *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pp. 82–89, Dublin, Ireland, 2014.
107. Ginter, F., J. Hajič, J. Luotolahti, M. Straka and D. Zeman, “CoNLL Shared Task - Automatically Annotated Raw Texts and Word Embeddings”, 2017, <https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-1989>, accessed in May 2018.
 108. Ballesteros, M., J. Herrera, V. Francisco and P. Gervás, “Are the Existing Training Corpora Unnecessarily Large?”, *Procesamiento del Lenguaje Natural*, Vol. 1, No. 48, pp. 21–27, 2012.
 109. Ballesteros, M., C. Dyer and N. A. Smith, “Improved Transition-Based Parsing by Modeling Characters instead of Words with LSTMs”, *Proceedings of the Conference on Empirical Methods in Natural Language (EMNLP)*, pp. 349–359, Lisbon, Portugal, 2015.
 110. Nivre, J., “Universal Dependencies Version 2.2”, 2018, <http://hdl.handle.net/11234/1-2837>, accessed in June 2018.
 111. Özateş, Ş. B., A. Özgür, T. Güngör and B. Öztürk, “A Morphology-Based Representation Model for LSTM-Based Dependency Parsing of Agglutinative Languages”, *Proceedings of the CoNLL Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pp. 238–247, Brussels, Belgium, 2018.
 112. Zeman, D., J. Hajič, M. Popel, M. Potthast, M. Straka, F. Ginter, J. Nivre and S. Petrov, “CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies”, *Proceedings of the CoNLL Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pp. 1–21, Brussels, Belgium, 2018.

113. Dependencies, U., “CoNLL-U Format”, 2014, <https://universaldependencies.org/format.html>, accessed in June 2018.
114. Özateş, Ş. B., “A Morphology-Based Representation Model for LSTM-based Dependency Parsing of Agglutinative Languages - Source Code”, 2018, <https://github.com/CoNLL-UD-2018/BOUN>, accessed in June 2018.
115. Moravcsik, E., *Introducing Language Typology*, Cambridge University Press, Cambridge, 2013.
116. Bojanowski, P., E. Grave, A. Joulin and T. Mikolov, “Enriching Word Vectors with Subword Information”, arXiv:1607.04606 [cs], 2016.
117. Dozat, T. and C. D. Manning, “Deep Biaffine Attention for Neural Dependency Parsing”, *Proceedings of the Fifth International Conference on Learning Representations (ICLR)*, Toulon, France, 2017.
118. Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, *The Journal of Machine Learning Research*, Vol. 15, No. 1, pp. 1929–1958, 2014.
119. Sudha, V. P. and R. Kowsalya, “A Survey on Deep Learning Techniques Applications And Challenges”, *International Journal of Advance Research In Science And Engineering (IJARSE)*, Vol. 4, No. 3, pp. 311–317, 2015.
120. Brill, E., “A Simple Rule-Based Part of Speech Tagger”, *Proceedings of the Third Conference on Applied Natural Language Processing*, p. 152–155, Trento, Italy, 1992.
121. Poria, S., E. Cambria, L.-W. Ku, C. Gui and A. Gelbukh, “A Rule-Based Approach to Aspect Extraction from Product Reviews”, *Proceedings of the Second Workshop on Natural Language Processing for Social Media*, pp. 28–37, Dublin, Ireland, 2014.

122. Oflazer, K., “Dependency Parsing with an Extended Finite-State Approach”, *Computational Linguistics*, Vol. 29, No. 4, pp. 515–544, 2003.
123. Sennrich, R., G. Schneider, M. Volk and M. Warin, “A New Hybrid Dependency Parser for German”, *Proceedings of the German Society for Computational Linguistics and Language Technology*, pp. 115–124, Potsdam, Germany, 2009.
124. Ramasamy, L. and Z. Žabokrtský, “Tamil Dependency Parsing: Results Using Rule Based and Corpus Based Approaches”, *Proceedings of the International Conference on Intelligent Text Processing and Computational Linguistics*, pp. 82–95, Berlin, Heidelberg, 2011.
125. Boguslavsky, I., L. Iomdin, V. Sizov, L. Tsinman and V. Petrochenkov, “Rule-Based Dependency Parser Refined by Empirical and Corpus Statistics”, *Proceedings of the International Conference on Dependency Linguistics*, pp. 318–327, Barcelona, Spain, 2011.
126. Korzeniowski, M. and J. Mazurkiewicz, “Rule based Dependency Parser for Polish Language”, *Proceedings of the International Conference on Artificial Intelligence and Soft Computing*, pp. 498–508, Zakopane, Poland, 2017.
127. Che, W., Y. Liu, Y. Wang, B. Zheng and T. Liu, “Towards Better UD parsing: Deep Contextualized Word Embeddings, Ensemble, and Treebank Concatenation”, *Proceedings of the CoNLL Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pp. 55–64, Brussels, Belgium, 2018.
128. Tsarfaty, R., D. Seddah, Y. Goldberg, S. Kuebler, Y. Versley, M. Candito, J. Foster, I. Rehbein and L. Tounsi, “Statistical Parsing of Morphologically Rich Languages (SPMRL) What, How and Whither”, *Proceedings of the First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pp. 1–12, Los Angeles, USA, 2010.

129. Zeman, D. and Z. Žabokrtský, “Improving Parsing Accuracy by Combining Diverse Dependency Parsers”, *Proceedings of the Ninth International Workshop on Parsing Technology*, pp. 171–178, Vancouver, Canada, 2005.
130. Ruppert, E., J. Klesy, M. Riedl and C. Biemann, “Rule-Based Dependency Parse Collapsing and Propagation for German and English”, *Proceedings of the German Society for Computational Linguistics and Language Technology*, pp. 58–66, Duisburg-Essen, Germany, 2015.
131. Ambati, B. R., S. Husain, J. Nivre and R. Sangal, “On the Role of Morphosyntactic Features in Hindi Dependency Parsing”, *Proceedings of the First Workshop on Statistical Parsing of Morphologically-Rich Languages*, p. 94–102, Los Angeles, USA, 2010.
132. Goldberg, Y. and M. Elhadad, “Easy-First Dependency Parsing of Modern Hebrew”, *Proceedings of the First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pp. 103–107, Los Angeles, USA, 2010.
133. Marton, Y., N. Habash and O. Rambow, “Improving Arabic Dependency Parsing with Lexical and Inflectional Morphological Features”, *Proceedings of the First Workshop on Statistical Parsing of Morphologically-Rich Languages*, pp. 13–21, Los Angeles, USA, 2010.
134. Habash, N. and O. Rambow, “Arabic Tokenization, Part-of-Speech Tagging and Morphological Disambiguation in One Fell Swoop”, *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 573–580, Ann Arbor, USA, 2005.
135. Vania, C., A. Grivas and A. Lopez, “What Do Character-level Models Learn About Morphology? The Case of Dependency Parsing”, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2573–2583, Brussels, Belgium, 2018.

136. Dehouck, M. and P. Denis, “Phylogenic Multi-Lingual Dependency Parsing”, *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pp. 192–203, Minneapolis, USA, 2019.
137. Eryiğit, G. and K. Oflazer, “Statistical Dependency Parsing for Turkish”, *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 89–96, Trento, Italy, 2006.
138. Eryiğit, G., T. İlbay and O. A. Can, “Multiword Expressions in Statistical Dependency Parsing”, *Proceedings of the Second Workshop on Statistical Parsing of Morphologically Rich Languages*, pp. 45–55, Dublin, Ireland, 2011.
139. Seeker, W. and Ö. Çetinoğlu, “A Graph-Based Lattice Dependency Parser for Joint Morphological Segmentation and Syntactic Analysis”, *Transactions of the Association for Computational Linguistics*, Vol. 3, pp. 359–373, 2015.
140. Öztürk, B., *Case, Referentiality and Phrase Structure*, Ph.D. Dissertation, Harvard University, 2004.
141. Keskin, C., *Subject Agreement-Dependency of Accusative Case in Turkish or Jump-Starting Grammatical Machinery*, Ph.D. Dissertation, Utrecht University, 2009.
142. Akkuş, F., “Light Verb Constructions in Turkish: A Case for DP Predication and Blocking”, *Proceedings of the 9th Workshop on Altaic Formal Linguistics (MITWPL)*, pp. 133–145, Cambridge, USA, 2013.
143. Solak, E., “Accusative Licensing of Nouns in Turkish”, *Proceedings of the Sixth Workshop on Turkic and Languages in Contact with Turkic*, Online, 2021.
144. Ramisch, C., S. R. Cordeiro, A. Savary, V. Vincze, V. Barbu Mititelu, A. Bhatia, M. Buljan, M. Candito, P. Gantar, V. Giouli, T. Güngör, A. Hawwari,

- U. Iñurrieta, J. Kovalevskaitė, S. Krek, T. Lichte, C. Liebeskind, J. Monti, C. Parra Escartín, B. QasemiZadeh, R. Ramisch, N. Schneider, I. Stoyanova, A. Vaidya and A. Walsh, “Edition 1.1 of the PARSEME Shared Task on Automatic Identification of Verbal Multiword Expressions”, *Proceedings of the Joint Workshop on Linguistic Annotation, Multiword Expressions and Constructions (LAW-MWE-CxG)*, pp. 222–240, Santa Fe, USA, 2018.
145. Akalın, Ş. H., R. Toparlı and B. Tezcan Aksu, *Atasözleri ve Deyimler Sözlüğü*, Türk Dil Kurumu, Ankara, 2009.
146. Kingma, D. P. and J. Ba, “Adam: A Method for Stochastic Optimization”, arXiv:1412.6980 [cs], 2014.
147. Kondratyuk, D. and M. Straka, “75 Languages, 1 Model: Parsing Universal Dependencies Universally”, *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the Ninth International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 2779–2795, Hong Kong, China, 2019.
148. Noreen, E. W., *Computer-Intensive Methods for Testing Hypotheses*, Wiley, New York, 1989.
149. Riezler, S. and J. T. Maxwell, “On Some Pitfalls in Automatic Evaluation and Significance Testing for MT”, *Proceedings of the Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pp. 57–64, Ann Arbor, USA, 2005.
150. Bhat, I., R. A. Bhat, M. Shrivastava and D. Sharma, “Joining Hands: Exploiting Monolingual Treebanks for Parsing of Code-mixing Data”, *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 324–330, Valencia, Spain, 2017.

151. Partanen, N., K. Lim, M. Rießler and T. Poibeau, “Dependency Parsing of Code-Switching Data with Cross-Lingual Feature Representations”, *Proceedings of the Fourth International Workshop on Computational Linguistics of Uralic Languages*, pp. 1–17, Helsinki, Finland, 2018.
152. Braggaar, A. and R. van der Goot, “Challenges in Annotating and Parsing Spoken, Code-switched, Frisian-Dutch Data”, *Proceedings of the Second Workshop on Domain Adaptation for Natural Language Processing*, pp. 50–58, Kyiv, Ukraine, 2021.
153. van der Goot, R., A. Üstün, A. Ramponi, I. Sharaf and B. Plank, “Massive Choice, Ample Tasks (MaChAmp): A Toolkit for Multi-task Learning in NLP”, *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 176–197, Online, 2021.
154. McClosky, D., E. Charniak and M. Johnson, “Effective Self-Training for Parsing”, *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL)*, pp. 152–159, New York, USA, 2006.
155. Rybak, P. and A. Wróblewska, “Semi-Supervised Neural System for Tagging, Parsing and Lemmatization”, *Proceedings of the CoNLL Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pp. 45–54, Brussels, Belgium, 2018.
156. Yu, X., N. T. Vu and J. Kuhn, “Ensemble Self-Training for Low-Resource Languages: Grapheme-to-Phoneme Conversion and Morphological Inflection”, *Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pp. 70–78, Online, 2020.
157. Rotman, G. and R. Reichart, “Deep Contextualized Self-Training for Low Resource Dependency Parsing”, *Transactions of the Association for Computational*

- Linguistics*, Vol. 7, pp. 695–713, 2019.
158. Bhat, I., R. A. Bhat, M. Shrivastava and D. Sharma, “Universal Dependency Parsing for Hindi-English Code-Switching”, *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pp. 987–998, New Orleans, USA, 2018.
 159. Lim, K. and T. Poibeau, “A System for Multilingual Dependency Parsing based on Bidirectional LSTM Feature Representations”, *Proceedings of the CoNLL Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pp. 63–70, Vancouver, Canada, 2017.
 160. Partanen, N., R. Blokland, K. Lim, T. Poibeau and M. Rießler, “The First Komi-Zyrian Universal Dependencies Treebanks”, *Proceedings of the Second Workshop on Universal Dependencies (UDW)*, pp. 126–132, Brussels, Belgium, 2018.
 161. Çetinoğlu, Ö. and Ç. Çöltekin, “Challenges of Annotating a Code-Switching Treebank”, *Proceedings of the 18th International Workshop on Treebanks and Linguistic Theories (TLT, SyntaxFest)*, pp. 82–90, Paris, France, 2019.
 162. Seddah, D., F. Essaidi, A. Fethi, M. Futeral, B. Muller, P. J. Ortiz Suárez, B. Sagot and A. Srivastava, “Building a User-Generated Content North-African Arabizi Treebank: Tackling Hell”, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 1139–1150, Online, 2020.
 163. Straka, M. and J. Straková, “Tokenizing, POS Tagging, Lemmatizing and Parsing UD 2.0 with UDPipe”, *Proceedings of the CoNLL Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pp. 88–99, Vancouver, Canada, 2017.
 164. Blei, D. M., A. Y. Ng and M. I. Jordan, “Latent Dirichlet Allocation”, *Journal*

of Machine Learning Research, Vol. 3, p. 993–1022, 2003.

165. Müller-Eberstein, M., R. van der Goot and B. Plank, “Genre as Weak Supervision for Cross-lingual Dependency Parsing”, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMLNP)*, pp. 4786–4802, Online and Punta Cana, Dominican Republic, 2021.
166. Sato, M., H. Manabe, H. Noji and Y. Matsumoto, “Adversarial Training for Cross-Domain Universal Dependency Parsing”, *Proceedings of the CoNLL Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pp. 71–79, Vancouver, Canada, 2017.
167. Jamatia, A., B. Gambäck and A. Das, “Part-of-Speech Tagging for Code-Mixed English-Hindi Twitter and Facebook Chat Messages”, *Proceedings of the International Conference Recent Advances in Natural Language Processing*, pp. 239–248, Hissar, Bulgaria, 2015.
168. Aguilar, G. and T. Solorio, “From English to Code-Switching: Transfer Learning with Strong Morphological Clues”, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 8033–8044, Online, 2020.
169. Dehouck, M. and P. Denis, “A Framework for Understanding the Role of Morphology in Universal Dependency Parsing”, *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2864–2870, Brussels, Belgium, 2018.
170. Sandhan, J., A. Krishna, A. Gupta, L. Behera and P. Goyal, “A Little Pretraining Goes a Long Way: A Case Study on Dependency Parsing Task for Low-resource Morphologically Rich Languages”, *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics (EACL): Student Research Workshop*, pp. 111–120, Online, 2021.

171. Li, Z., W. Che and T. Liu, “Improving Dependency Parsing Using Punctuation”, *Proceedings of the International Conference on Asian Language Processing*, pp. 53–56, Harbin, China, 2010.
172. Spitkovsky, V. I., H. Alshawi and D. Jurafsky, “Punctuation: Making a Point in Unsupervised Dependency Parsing”, *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, pp. 19–28, Portland, USA, 2011.
173. Sanh, V., L. Debut, J. Chaumond and T. Wolf, “DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter”, arXiv:1910.01108 [cs], 2019.
174. Liu, Y., M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer and V. Stoyanov, “RoBERTa: A Robustly Optimized BERT Pretraining Approach”, arXiv:1907.11692 [cs], 2019.
175. Grünwald, S., A. Friedrich and J. Kuhn, “Applying Occam’s Razor to Transformer-Based Dependency Parsing: What Works, What Doesn’t, and What is Really Necessary”, *Proceedings of the 17th International Conference on Parsing Technologies and the IWPT Shared Task on Parsing into Enhanced Universal Dependencies (IWPT)*, pp. 131–144, Online, 2021.
176. Das, A. and B. Gambäck, “Identifying Languages at the Word Level in Code-Mixed Indian Social Media Text”, *Proceedings of the 11th International Conference on Natural Language Processing*, pp. 378–387, Goa, India, 2014.
177. Arkhangelskiy, T., “Komi-Zyrian Corpora”, 2019, <http://komi-zyrian.web-corpora.net/>, accessed in December 2021.
178. Arkhangelskiy, T., “Corpora of Social Media in Minority Uralic Languages”, *Proceedings of the Fifth International Workshop on Computational Linguistics for Uralic Languages*, pp. 125–140, Tartu, Estonia, 2019.
179. Aguilar, G., S. Kar and T. Solorio, “LinCE: A Centralized Benchmark for Lin-

- guistic Code-switching Evaluation”, *Proceedings of The 12th Language Resources and Evaluation Conference (LREC)*, pp. 1803–1813, Marseille, France, 2020.
180. Mave, D., S. Maharjan and T. Solorio, “Language Identification and Analysis of Code-Switched Social Media Text”, *Proceedings of the Third Workshop on Computational Approaches to Linguistic Code-Switching*, pp. 51–61, Melbourne, Australia, 2018.
 181. Singh, K., I. Sen and P. Kumaraguru, “Language Identification and Named Entity Recognition in Hinglish Code Mixed Tweets”, *Proceedings of The 56th Annual Meeting of the Association for Computational Linguistics (ACL): Student Research Workshop*, pp. 52–58, Melbourne, Australia, 2018.
 182. Singh, K., I. Sen and P. Kumaraguru, “A Twitter Corpus for Hindi-English Code Mixed POS Tagging”, *Proceedings of the Sixth International Workshop on Natural Language Processing for Social Media*, pp. 12–17, Melbourne, Australia, 2018.
 183. Yılmaz, E., M. Andringa, S. Kingma, J. Dijkstra, F. van der Kuip, H. Van de Velde, F. Kampstra, J. Algra, H. van den Heuvel and D. van Leeuwen, “A Longitudinal Bilingual Frisian-Dutch Radio Broadcast Database Designed for Code-Switching Research”, *Proceedings of the Tenth International Conference on Language Resources and Evaluation*, pp. 4666–4669, Portorož, Slovenia, 2016.
 184. Treffers-Daller, J., “Turkish-German Code-Switching Patterns Revisited: What Naturalistic Data Can(not) Tell Us”, *Advances in Contact Linguistics: In honour of Pieter Muysken*, Vol. 57, p. 237, 2020.
 185. Ma, X., Z. Hu, J. Liu, N. Peng, G. Neubig and E. Hovy, “Stack-Pointer Networks for Dependency Parsing”, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pp. 1403–1414, Melbourne, Australia, 2018.

186. Grave, E., P. Bojanowski, P. Gupta, A. Joulin and T. Mikolov, “Learning Word Vectors for 157 Languages”, *Proceedings of the 11th International Conference on Language Resources and Evaluation (LREC)*, Miyazaki, Japan, 2018.
187. Loshchilov, I. and F. Hutter, “Decoupled Weight Decay Regularization”, *Proceedings of the Seventh International Conference on Learning Representations (ICLR)*, New Orleans, USA, 2019.
188. Tyers, F. M., M. Sheyanova and J. N. Washington, “UD Annotatrix: An Annotation Tool for Universal Dependencies”, *Proceedings of the 16th International Workshop on Treebanks and Linguistic Theories*, pp. 10–17, Prague, Czech Republic, 2017.
189. Heinecke, J., “ConlluEditor: A Fully Graphical Editor for Universal Dependencies Treebank Files”, *Proceedings of the Third Workshop on Universal Dependencies (UDW, SyntaxFest)*, pp. 87–93, Paris, France, 2019.
190. Yıldız, O. T., E. Solak, Ş. Çandır, R. Ehsani and O. Görgün, “Constructing a Turkish Constituency Parse Treebank”, *Proceedings of 30th International Symposium on Computer and Information Sciences*, pp. 339–347, London, UK, 2016.
191. Eryiğit, G., “ITU Treebank Annotation Tool”, *Proceedings of the Linguistic Annotation Workshop*, pp. 117–120, Prague, Czech Republic, 2007.
192. Stenetorp, P., S. Pyysalo, G. Topić, T. Ohta, S. Ananiadou and J. Tsujii, “Brat: A Web-Based Tool for NLP-Assisted Text Annotation”, *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pp. 102–107, Avignon, France, 2012.
193. Popel, M., Z. Žabokrtský and M. Vojtek, “Udapi: Universal API for Universal Dependencies”, *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW)*, pp. 96–101, Gothenburg, Sweden, 2017.

194. Zeman, D., “Universal Dependencies Tools”, 2014, <https://github.com/universaldependencies/tools>, accessed in April 2022.
195. Dependencies, U., “CoNLL-U Plus Format”, 2014, <https://universaldependencies.org/ext-format.html>, accessed in January 2020.