# JOINT LEARNING OF SYNTAX AND ARGUMENT STRUCTURE IN DEPENDENCY PARSING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS OF
THE MIDDLE EAST TECHNICAL UNIVERSITY
BY

TOLGA KAYADELEN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE
OF
DOCTOR OF PHILOSOPHY
IN
THE DEPARTMENT OF COGNITIVE SCIENCE

FEBRUARY, 2023

**JOINT LEARNING OF SYNTAX AND ARGUMENT STRUCTURE IN DEPENDENCY PARSING**

submitted by **TOLGA KAYADELEN** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Cognitive Science Department, Middle East Technical University** by,

Prof. Dr. Banu Günel Kılıç
Dean, **Graduate School of Informatics** _____

Dr. Ceyhan Temürcü
Head of Department, **Cognitive Science** _____

Prof. Dr. Cem Bozşahin
Supervisor, **Cognitive Science, METU** _____

**Examining Committee Members:**

Assoc. Prof. Dr. Barbaros Yet
Cognitive Science, METU _____

Prof. Dr. Cem Bozşahin
Cognitive Science, METU _____

Assist. Prof. Dr. Burcu Can Buğlalılar
Computing Science and Mathematics, University of Stirling _____

Prof. Dr. Balkız Öztürk
Linguistics, Boğaziçi University _____

Prof. Dr. Deniz Zeyrek Bozşahin
Cognitive Science, METU _____

**Date: 27.02.2023**

**I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.**

**Name, Surname: Tolga Kayadelen**

**Signature :**

# ABSTRACT

## JOINT LEARNING OF SYNTAX AND ARGUMENT STRUCTURE IN DEPENDENCY PARSING

Kayadelen, Tolga
Ph.D., Department of Cognitive Science
Supervisor: Prof. Dr. Cem Bozşahin

February, 2023, 92 pages

This thesis is an experimentation on learning predicate-argument structure and syntax within the dependency parsing framework. The linguistic representation used in this framework is dependency grammar. In dependency grammar, the predicate argument structure of a sentence is represented in the form of labeled and uni-directed dependency trees.

Dependency parsing is the problem of inducing a dependency grammar from data. The dependency parsing problem can be conceived of as a combination of two tasks: head-selection (arc-prediction) and label-classification. Head selection aims to determine head-modifier relations in the sentence by associating modifiers with the heads that they modify using dependency arcs. On the other hand, label classification aims to determine the grammatical role of each word in the sentence. In existing parsing approaches, these two tasks are usually stacked on top of one another where the former takes precedence over the latter. In other words, models first try to predict the dependency arcs by connecting dependents to their heads and generating an unlabeled tree, following which they assign labels to the arcs of the tree. In this set up, dependency labeling have no impact at all in predicting the correct dependency tree as it applies only after the tree is already generated.

In this study, instead of generating an unlabeled dependency tree and then using dependency labels only as names over the arcs in that tree, we give dependency labels a more central role in the overall parsing process. We first predict the dependency

label of each word, therefore predicting its grammatical role in the sentence, and then generate the dependency tree based on those predictions. We call this method *label-first* parsing. As it will be shown, this approach improves the parsing accuracy considerably for a number of languages.

Another important aspect of the *label-first* parsing approach is that in this approach syntactic attachment is mainly driven by the argument structure that the system detects, therefore a lot of weight is put on predicting the predicates and the arguments correctly. We experiment with a variety of languages and show that a parser that can accurately predict the predicate and argument roles early in the parsing process can perform better across a number of languages compared to one which does not. Comparing the variation in parsing performance across languages, and considering their typological characteristics, we also try to derive conclusions about the suitability of the dependency representation for learning the predicate-argument structure in languages with different linguistic properties.

Keywords: Dependency parsing, language processing, syntax, argument structure, deep learning

# ÖZ

## ÜYE YAPISI VE BAĞLILIK ÇÖZÜMLEMESİNİN BİRLİKTE ÖĞRENİMİ

Kayadelen, Tolga
Doktora, Bilişsel Bilimler Bölümü
Tez Yöneticisi: Prof. Dr. Cem Bozşahin

2023, 92 sayfa

Bağlılık ayrıştırması bir tümcedeki bağlılık ilişkilerini saptayarak bunların bağlılık türlerini sınıflandırma işlemidir. Bağlılık ilişkilerinin saptanması, tümcedeki iye-uydu (head-dependent) yapısının çözümlenmesi anlamına gelir. Bağlılık ayrıştırıcılar, tümce içindeki iye-uydu ilişkilerini saptamanın yanı sıra, bu ilişkierin dilbilgisel türlerini de sınıflandıran yapay zeka modelleridir. Burada bahsedilen dilbilgisel ilişkiler, tümcenin üye-yapısını tanımlayan *özne*, *nesne*, *niteleyici* gibi dilbilgisel rolleri kapsar. Bu bağlamda, bağlılık türlerinin sınıflandırılması, bir başka deyişle *bağlılık etiketleme* tümcedeki her sözcüğün tümcenin üye yapısındaki görevinin tanımlanması olarak yorumlanabilir.

Bu çalışmada, mevcut bağlılık ayrıştırıcıların aksine, tümcede öncelikle üye rollerini ve üye yapısını saptamaya çalışan bir ayrıştırma modeli geliştirilmiştir. Bu model bir çok farklı dile ve veri setine uygulanmış ve modelin belirli dillerde bağlılık ayrıştırma başarımında önemli artışlara yol açtığı saptanmıştır. Geliştirilen modelin farklı dillerdeki başarımı göz önünde bulundurularak, dilbilgisel bilgiyi bağlılık ilişkileri olarak kodlayan yaklaşımın ve veri setlerinin, farklı dillerde üye yapısını bu tarz bir veri üzerinden öğrenmeyi ne kadar mümkün kılıp kılmadığına dair gözlemler sunulmuştur.

Anahtar Kelimeler: Bağlılık analizi, bağlılık etiketleme, üye yapısı, doğal dil işleme, derin öğrenme

To Ada

## ACKNOWLEDGMENTS

It has been so long since the start of this journey that I cannot believe I am at this stage writing the acknowledgements, which means the journey is coming to an end. I am grateful to a lot of people who made it possible.

First and foremost, I would like to thank my supervisor, Cem Bozşahin, for always believing in me and supporting me. Overwhelmed by work-related duties, I know I have not been the best student at times. However, his constant patience and trust in me have always kept me going. It is hard to describe how much I have learned from working with him, both during and before the writing of this thesis as my supervisor and my teacher.

Special thanks to Deniz Zeyrek Bozşahin for her constant support and encouragement. Her directions during my years as a graduate student at the Cognitive Science department have had enormous impact on my career. In terms of the knowledge and skills that I have acquired during these years, I am a completely different person than I was when I first joined the department, and she played a big role in this.

I would like to thank Burcu Can Buğlalılar for her insights and guidance about dependency parsing which ensured that the research presented in this thesis is going in the right direction and asking the correct questions.

Thanks to my additional jury members, Balkız Öztürk and Barbaros Yet, for their insightful comments and questions in terms of the linguistic and machine learning aspects of this thesis. Their valuable input has helped me shape the thesis to its final form.

I have worked on this thesis while being employed full-time at Google for the past 7 years. It is a difficult undertaking to complete a PhD thesis while working full-time in a demanding job, and the last few weeks towards completion have been particularly tiring and stressful for a various reasons. Special thanks to Sabine Lehmann, who was my manager at that time, for her support and encouragement which helped me get through those difficult times.

I would like to thank my dearest friend, Adnan Öztürel, for his help with machine learning questions and for his constant support.

Thanks to my beloved family, Saadet Apaydın, Esra Kayadelen, Ebru Doğar and Nusret Doğar for always being there for me. Biggest thanks to my wife, Buse Kayadelen, for her friendship and tenderness, and for bearing with me during all these years I have been working on this thesis.

Finally, thank you to my lovely daughter, Ada – who is 1,5 years old at the time I am writing these words – for being the joy of my life. You have made all the difference.

## TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

## LIST OF ABBREVIATIONS

| | |
|---|---|
| 1sg | First person singular agreement suffix |
| 2sg | Second person singular agreement suffix |
| 3sg | Third person singular agreement suffix |
| AAD | Argument Adjunct Distinction |
| acc | Accusative Case |
| AMR | Abstract Meaning Representation |
| aor | Aorist suffix |
| Aggl | Agglutinative |
| BERT | Birectional Encoder Representations from Transformers |
| c-structure | Constituent Structure |
| en | English |
| CCG | Combinatory Categorial Grammar |
| CFG | Context Free Grammar |
| CLE | Chu-Liu-Edmonds |
| CoNNL | Conference on Computational Natural Language Learning |
| de | German |
| dat | Dative case |
| DNN | Deep Neural Network |
| DQN | Deep Q Network |
| ECM | Exceptional Case Marking |
| ERG | Ergative |
| FFNN | Feed forward Neural Network |

| | |
|---|---|
| fi | Finnish |
| gen | Genitive marker |
| ko | Korean |
| LAS | Labeled Attachment Score |
| LCS | Lexical Conceptual Structure |
| LFG | Lexical Functional Grammar |
| LLM | Large language models |
| loc | Locative case |
| LSTM | Long Short-term Memory |
| f-structure | Functional Structure |
| MDP | Markov Decision Process |
| ML | Machine Learning |
| MLP | Multi layered Perceptron |
| MST | Maximum Spanning Tree |
| neg | Negation suffix |
| NLP | Natural Language Processing |
| NP | Noun Phrase |
| PALM | Pathways Language Model |
| past | Past tense |
| pres | Present tense |
| Propbank | Proposition Bank |
| PoS | Part of Speech |
| poss | Possessive marker |
| PP | Prepositional Phrase |
| PUD | Parallel Universal Dependencies |
| ReLU | Rectified Linear Unit |

| | |
|---|---|
| RL | Reinforcement Learning |
| RNN | Recurrent Neural Network |
| ru | Russian |
| SD | Stanford Dependencies |
| SRL | Semantic Role Labeling |
| SVM | Support Vector Machine |
| tr | Turkish |
| UAS | Unlabeled Attachment Score |
| UD | Universal Dependencies |
| VN | Verbal Noun |
| VP | Verb phrase |
| zh | Chinese |

# CHAPTER 1

# INTRODUCTION

The central issue in this thesis is learning predicate argument structure within a dependency parsing framework using data driven machine learning models. Parsing is an important domain of application in Natural Language Processing (NLP) for a couple of reasons. First, we know that natural language syntax, in and of itself, is a complex system due to many subtleties that natural languages possess. Second, it is a system that interfaces with other systems such as morphology, semantics, and even prosody, all of which might impose well-formedness restrictions on it. Such level of complexity provided by natural language syntax makes it one of the most interesting domains to implement and test parsers for and to see, to what extent, even the most powerful statistical models can achieve the same success that a toddler achieves in understanding complex sentences. Second, from a purely practical perspective, dependency parsing has proven to be a useful first task for other downstream applications such as semantic role labeling (Marcheggiani et al., 2017) [1], relation extraction (Zhang et al., 2018) [2] and even machine translation (Chen et al., 2017) [3].

Achieving high performance results in any machine learning problem has at least three fundamental requirements: a) a proper understanding and representation of the phenomenon one is dealing with, b) an appropriate statistical model that can address the complexities of the phenomenon, and c) high quality data that is consistently annotated and has good representational coverage of the phenomenon. It is hard to compromise from any one of these without harming the others. That is to say, low quality in one of these components can easily nullify the quality of others. For example, unless one has a good mental model of the problem and of its representation, it is hard to create high quality data that is appropriate for the task. As a result, no matter how powerful a machine learning model one might employ, results will suffer as the data will not be representing the problem well enough. On the other hand, assume we have a good representation of the problem and very strong ML models. We still need to make sure that our dataset which the model will be trained on covers different distributional properties of the data and is annotated consistently. Lack of proper sampling will cause the model perform poorly in the face of new data, lack of consistent annotations will make it unable to learn the solution to the task as much as it potentially can.

Over the years, advances in deep learning and related developments in large language models (LLM) such as BERT (Devlin et al., 2019) [4] and PALM (Chowdhery et al. 2022) [5] has made it possible to gain better results in more and more NLP tasks.

Dependency parsing is no exception: recently developed models that rely on deep learning methods and large language models have achieved state of the art performance on a large number of languages easily, e.g. (Kondratyuk and Straka, 2019) [6]. In parallel, especially due to the success of Universal Dependencies (UD)[7] project in providing a unified way to annotate multilingual dependency treebanks, the amount of data available for dependency parsing has increased considerably in the past 10 years.

Of course, an increase in the number of treebanks covering different languages is not by itself a guarantee of quality parity between languages in terms of parsing performance. One reason for the lack of quality parity might be that the unified representation that UD imposes could be less suitable to languages with different linguistic properties. This is an inevitable risk that any unification efforts naturally bear and even though UD makes all effort to ensure that the annotation guidelines are sensitive to peculiarities of typologically different languages, the question of how suitable UD annotation scheme is to all the languages it aspires to cover is an open debate. Another issue that prevents quality parity is that there is less amount of annotated treebanks available for some languages. Complex ML models require large amounts of data to train on, and training on insufficient data easily leads to overfitting. Furthermore, certain languages are categorized as more challenging for dependency parsing than others. Some examples are morphologically rich languages that encode syntactic dependencies in morphological units and, accordingly, free word order languages that can assemble syntactic units in many different ways therefore increasing the amount of permutations a parser has learn. This brings us back to the importance of data for ML models. None of these linguistic properties are inherently problematic as long as one can have enough data to account for the additional complexity they bring to the parsing problem. However, there is no easy way to know beforehand what size of data is enough to accurately represent what type of language.

Turkish is a notable example of languages that are traditionally classified as challenging for dependency parsing (Oflazer, 2014) [8], both because of its morphological complexity and because it lacked sufficiently large dependency treebanks for many years. Until recently, the only available treebank for Turkish was IMST-UD (Sulubacak et al, 2016) [9] which was relatively small, consisting of 5.67k sentences. In recent years, this issue is addressed with some newly published treebanks such as the BOUN treebank (Turk et al., 2022) [10], Grammar Book Treebank (Coltekin, 2017) [11], both of which use UD annotation scheme, and Turkish Web Treebank (Kayadelen et al, 2020) [12] which uses a non-UD annotation scheme. Given these new treebanks, it is hard to categorize Turkish as an underrepresented language for dependency parsing any more. On the other hand, Turkish still consistently performs at the lower end of languages in shared tasks such as Zeman et al (2017) [13], which means either that current parsers are still far away from addressing its linguistic complexities even with the available data or that the data, in the way it is annotated, does not allow for capturing the peculiarities of the language.

This thesis experiments with a set of models that construes dependency parsing as a problem of optimizing predicate-argument structure. Dependency parsing can be understood as a combination of two tasks: a head selection task identifying the head-

dependent relations in the sentence, and a label prediction task identifying the nature of the grammatical relation that holds between the heads and dependents. In this thesis, while we preserve this two-level understanding of the parsing problem, we differ from the current state of the art parsers in that we interpret dependency labels as categorical markers defining the grammatical role (or function) each word plays in the overall predicate argument structure of the sentence. In this approach, dependency labels are conceptualized as properties predicated over words (rather than over dependency arcs), representing the words as functions seeking their arguments. For example, an *nsubj* dependency label represents a word as a function seeking the verbal predicate it takes as an argument, grouping it together with other dependency labels such as *dobj*, *iobj* or *ccomp* in UD label inventory. A *det* relation represents a word as a function seeking its nominal argument, similar to other labels such as *amod*, *nummod*, or *nmod*.

Based on the viewpoint described above, we develop a model which aims to predict the grammatical role of each word first, therefore trying to account for the predicates, arguments, and non-arguments in the sentence as a first step, before building the syntactic tree. From a modeling perspective, this approach is in contrast with the current state of the art parser implementations. As we review further in Chapter 3, in most of the current parser implementations the head-selection and label-prediction tasks are usually set up in a way that parsers first predict the dependency arcs connecting dependents to their heads (head selection), which are then labeled using one of the dependency labels. In that kind of set up, accurate identification of dependency labels, or in linguistic terms the grammatical functions, has no impact at all on how the syntactic tree is constructed. We call these mainstream approaches *head-first* to contrast them with the current methodology. In contrast to these approaches, the models developed in this thesis are *label-first* as they aim to identify the grammatical role that each word plays in the predicate argument structure of the sentence before generating the dependency tree.

The linguistic motivation behind the *label-first* parsing approach is that synctactic structure is mainly determined by predicate-argument structure, and in a parsing framework where predicates and arguments (or non-arguments) can be identified accurately, identification of dependency relations should follow almost trivially. Experimenting with a variety of languages and datasets, in Chapter 5 we show that the *label-first* parsing approach can indeed improve parsing accuracy (both labeled and unlabeled attachment score) significantly in a certain subset of these languages and datasets.

Analyzing the variability in parsing performance between the different languages experimented with, we then try to explain why the proposed approach works well only for certain languages but not for others. We show that this is mainly due to two reasons: a) the typological characteristics of the different languages experimented with, b) the lack of a typing mechanism in dependency grammars which defines the semantic and syntactic constraints (such as valency, subcategorization, and semantic selectional restrictions) over functional categories (e.g. predicates) in the linguistic representation. As we discuss and show with examples, the combined effect of these two facts is that learning predicate-argument structure from dependency treebanks is

tightly limited to the structures that a model can only explicitly see in the treebank. In languages that are more analytical in nature, the treebanks are less sparse in terms of different argument structure frames a model can witness a verb occur in, because in these langauges grammatical roles are overtly represented in the surface representation. On the other hand, in languages that are more on the synthetic/agglutinative side, grammatical roles can be covert in the surface representation due to rich morphological marking but can only be understood at the level of semantic representation. In turn, the lack of any lexical semantic information on predicates to explain unseen data in the linguistic representation hinders a successful induction of predicate argument structure in such languages more severely than others.

In terms its linguistic motivation, this thesis can be thought of as an effort which tests to what extent predicate argument structure is learnable in languages with different linguistic properties within a dependency parsing framework. The results that it reports have implications on how grammatical properties relevant to predicate argument structure should (or should not) be represented in linguistic frameworks that aim develop sufficiently rich datasets for typologically diverse languages. The main conclusion that it arrives is that the poorly typed representation that dependency formalism employs is problematic for learning predicate-argument structure accurately for all languages, and different languages suffer from this problem to different degrees.

## 1.1 Organization of the Thesis

The rest of this thesis is organized as follows.

In Chapter 2, we review the linguistic background of this thesis and provide an overview of the different approaches to predicate-argument structure.

In Chapter 3, we discuss the computational foundations of this thesis, reviewing graph-based and transition based dependency parsing algorithms. This is followed by a review of the recent state of the art neural parser implementations.

In Chapter 4, we review the properties of the data that we use for experiments in chapters 5 and 6. Specifically, we provide a linguistic analysis of the UD, focusing, where relevant, on the problematic aspects of it with respect to capturing important distinctions about predicate-argument structure.

In Chapter 5, we run a set of multilingual experiments and show that the *label first parsing* strategy, which aims to account for predicate and argument roles early in parsing stage, improves parsing accuracy across multiple languages and datasets. The results confirm our hypothesis that a parser which can resolve predicate argument structure early can perform better in resolving the syntactic dependencies. We then analyze why certain languages, but not others, seem to be benefiting more from this proposed strategy and derive conclusions about the learnability of predicate-argument

structure in different languages using dependency grammar as a linguistic representation.

In Chapter 6, we augment the model in chapter 5 with semantic role labels and perform label first parsing based on them. More specifically, we start parsing from annotated Semantic Role Labels (SRL), based on which we predict grammatical roles, on top of which we build the dependency tree. The main idea of this chapter is to test whether semantic role labels can further help a dependency parser in learning distinctions relevant to predicate-argument structure (e.g. the argument-adjunct distinction).

Chapter 7 provides results of a preliminary experiment which is much less conclusive and much more relevant for future work than any others in this thesis. We report an approach which tries to optimize for grammatical roles in a label-first parsing setting using a hybrid learning methodology, combining Reinforcement Learning (RL) algorithms with Supervised Learning. We explain the motivating idea behind this approach and review the relevant design employed.

Chapter 8 concludes the thesis and suggests some future work that can follow this thesis.

## 1.2 Contributions of the Thesis

The thesis has the following computational and linguistic contributions to the field of dependency parsing.

- It introduces a novel dependency parsing strategy which is different from the current implementation of the available state of the art parsers.
- It improves the state of the art metrics (Unlabeled Attachment Score (UAS) and Labeled Attachment Score (LAS)) for dependency parsing across multiple languages and UD datasets.
- It implements a BERT based dependency label classifier, illustrating the multilingual BERT models' performance on detecting grammatical roles without using any syntactic features (simply relying on word order and surface forms of words) in multiple languages.
- It develops an experiment that aims to resolve grammatical roles and syntactic dependencies in a hierarchical set up based on semantic role labels, which shows that using semantic role labels not only improve parsing accuracy, but also helps reduce predicate-argument structure errors with respect to arguments and adjuncts.
- From a linguistic point of view, it discusses that the dependency representation as adopted in UD, with its impoverished approach to representing linguistic properties of predicates, is not a sufficient representation for learning predicate-argument structure from data, especially in languages with certain typological characteristics.

# CHAPTER 2

# APPROACHES TO PREDICATE ARGUMENT STRUCTURE

Following Clark et al (2002) [14], predicate argument structure can be defined as "the dependencies that hold between lexical functor categories and their arguments". The common understanding in linguistics is that predicate-argument structure lies at the heart of syntax-semantics interface and is defined to a great extent by the lexical semantics of verbs. A great example of this idea has been shown in the seminal works of Talmy (1985, 1991) [15], [16].

In his works on cross-linguistic lexicalization patterns, Talmy (1985, 1991) discusses how languages with different typologies map semantic participants of *motion events* to argument or adjunct roles. Motion events are events that express movement along a trajectory, expressed by verbs such as *push, kick, roll, swing, move, run* and so on. A motion event can have five semantic components: a) a Figure, which is the entity that moves; a Path, which encodes the trajectory of movement; a Ground, which is the location with respect to which movement happens, a Manner component, which encodes how the motion happens; and finally the Motion itself which is expressed by the verb. The *path* component is central to any motion event, and should be lexicalized somehow in the surface syntax. Based on this observation, Talmy categorizes languages into two classes: those that encode the *path* semantics in the verb meaning (verb-framed languages) and those that can realize them in non-verbal categories such as adpositions (satellite-framed languages). This distinction helps explain how surface realization of other semantic components (such as Manner) of a motion event happens. In satellite-framed languages, since *path* is encoded in the adposition, the verb can encode motion+manner at the same time. Therefore, these languages have a lot of verbs that express manner of motion. On the other hand, in verb-framed languages, since since path is encoded in the verb, manner cannot be encoded in the verb as well. As a result, these languages have to employ adjuncts to express manner of motion.

1. [The pencil]$_{figure}$ [rolled]$_{motion+manner}$ [off]$_{path}$ [the table]$_{ground}$.
2. [Kalem]$_{figure}$ [masadan]$_{ground}$ [yuvarlanarak]$_{manner}$ [düştü]$_{motion+path}$.

An example of the mapping generalizations proposed by Talmy is exemplified in examples (1) and (2). Turkish, being a verb-framed language, employs an adjunct to express the manner of motion for a motion event while English, a satellite-framed language, can encode it in the verb and does not need an adjunct.

Talmy's observations constitute a great example of how semantics interacts with syntax, specifically how lexical semantics of verbs determine the predicate-argument structure of the verb in typologically different languages. A large amount of linguistic studies have dealt with formalizing this interface as part of a theory of grammar and different accounts have been proposed based on the linguistic theory one assumes. In this chapter, we review some of these approaches below.

## 2.1 Lexical-Projectionist View

According to Government and Binding (GB) theory (Chomsky, 1981)[17], linguistic competence consists of three basic components: lexical knowledge, PS-rules, and transformations. A matter of debate in this approach has always been the division of labor between these three components, especially between lexicon and syntax. More specifically, the question has been how to precisely define the nature of grammatical information that comes from the lexical component. Usually, two kind of lexical information has been recognized: the unstructured, "encyclopedic" information which has no relevance to the "computational system", and the systematic, grammatically relevant information that interfaces with the grammatical module and has the power to impose well-formedness conditions on grammar (Ramchand, 2008) [18]. The latter kind of information is assumed to be a proper subpart of the former, e.g. as Pinker (1989: 166)[19] notes "[...] there is a set of semantic elements and relations that is much smaller than the cognitively available [...] distinctions, and verb meanings are organized around them".

Chomsky's "Remarks" (1970) [20] can be taken as a milestone for the so called lexical-projectionist view of semantics-syntax interface. The idea put forward there was that lexical items project into syntax argument structural information that is stored in them. This idea has assumed a multi-layered representation of the grammatical system, where one level of representation (the lexicon) and the other one (the syntax) conspire to determine grammaticality. This idea is later taken up to its full potential in works such as Fillmore (1970)[21] and Levin (1993) [22] and related work). More explicitly, the common understanding in works of this clay has been that different patterns of behavior exhibited by verbs could be explained by taking into consideration the semantic role lists they define in their lexical conceptual structure.

Given the semantic role lists which are defined in the lexicon, these multi-stratal accounts of grammar would have to also define a theory of "linking" governing how the semantic information gets mapped onto syntax. Various linking theories have been proposed, and in most of them the central idea has been that the linking of semantic information should observe a thematic hierarchy of some sort. The thematic hierarchy proposed by Fillmore (1968) is shown below:

3. Fillmore (1968) Thematic Hierarchy:

   Agent > Experiencer > Instrument > Theme/Patient

The linking (or mapping) operates by proposing generalizations for associating semantic roles in the hierarchies like the ones in (3) with grammatical role hierarchies like (subject>object ...). This is done by positing certain mapping rules. For example, the subject selection rule states that the argument of a verb which bears the highest ranked semantic role is mapped to the highest ranked grammatical role in the syntactic role hierarchy. Interpreted from this perspective, Fillmore's hierarchy entails that if there is an Agent semantic role in the verb's lexical entry then it has to be mapped to the Subject grammatical role, if not an Experiencer is mapped to the subject role. If none of these roles exist, then the Subject is an Instrument or else, it is Objective. To illustrate this mapping in action, we can consider Fillmore's (1968) examples in his famous work *'The grammar of hitting and breaking'*. The following examples are adapted from that work:

4. a. The door opened. (Subject=Theme)
   b. John opened the door. (Subject=Agent, Object=Theme)
   c. A key opened the door. (Subject=Instrument)
   d. *A key opened the door by John. *(Subject=Instrument, Oblique=Agent)
5. a. The fence broke. (Subject=Theme)
   b. A stick broke the fence. (Subject=Instrument, Object=Theme)
   c. John broke the fence with a stick. (Subject=Agent, Oblique=Instrument)
   d. *The fence broke with a stick. *(Subject=Theme, Oblique=Instrument)

The sentences (4d) and (5d) are ungrammatical because they violate the subject selection principle based on the thematic role hierarchy proposed in (3), which does not allow for selecting an Instrument subject when there is an Agent present (4d) or a Theme subject when there is an Instrument present (5d).

There has been some dissatisfaction concerning the thematic role approach to syntax-semantics interface, which has mainly been related to the definition of thematic role labels. It has been acknowledged that some verbs require finer-grained thematic roles since the inventory of thematic roles do not seem to adequately capture the semantic properties of arguments projected by these verbs. Some of these verbs are *require*, *confirm*, *contemplate*, and *facilitate* which, as noted by Levin and Rappoport-Hovav (2005: 40)[23], ask for individually tailored semantic roles. However, as also noted by the same researchers, "positing such roles could result in a staggering number of roles; furthermore, little is gained by introducing them since many would not enter into any significant linguistic generalizations". Moreover, once one starts reducing the grain-size of the thematic roles and decomposing them into smaller units, there is not a principled reason to stop at any particular point. One can go on and on as long as the need arises.

This has led researchers like Dowty (1990)[24] to give up the idea of identifying thematic roles as primitives in a theory of argument structure altogether. Arguing that thematic hierarchies do not have the explanatory potential that they are assumed to

have, Dowty (1990) suggested thematic roles are derived and probabilistically determined based on the number of entailments that hold between the verb and its arguments. That quite innovative view captured and explained many problems related to argument selection/realization. More striking was Dowty's conception of semantic roles as being partly derived, rather than being determined solely by the lexemes. This idea has later on yielded itself into a whole paradigm of deriving semantic roles from syntactic patterns, and refusing the lexemes any argument structural information altogether; i.e. the "constructivist approaches".

### 2.2 Construction Grammar (CxG)

In a constructionish view of grammar (Goldberg, 1995[25]; Michaelis, 2006 [26] among others), lexemes contain no semantic (and in fact syntactic) information at all; they are just packages of grammatically irrelevant "encyclopedic" content. According to CxG, there is an inventory of syntactically represented meanings that exist independently of verbs, and idiosyncratic semantic requirements of verbs use one of these syntactic representations, allowing or disallowing the verbs to be inserted within one constructional frame but not the other.

These approaches usually base their claim on examples where it seems that the categorical information related with the verb is overridden as a result of the verb's appearing in a variety of non-canonical usages that it should not normally be expected to appear. As example, we can consider Goldberg's (2003: 592)[27] famous discussion of verbs "laugh" and "sneeze". In a framework where verbs include categorical information, these verbs would be categorized one-place predicates assigning an agent role to their arguments.

6. a. We laughed.
   b. John sneezed.

On the other hand, Goldberg notes that these verbs could easily be used in a ditransitive construction, as in

7. a. We laughed our conversation to an end.
   b. John sneezed his tooth right across town.

Now, we know that the ditransitive pattern usually expresses events of transfer and caused- motion. Verbs like "give", "wipe", "drag", "push" are commonly cited members of this class of verbs (Diesel, 2004: 17):

8. a. She dragged the child into the car.
   b. He wiped the mud off his shoes.
   c. He pushed the book down the clute.

Comparing (7) to (8), CxG grammarians argue that it is implausible to assign the verbs "sneeze" and "laugh" a "caused-motion" sense besides their regular meaning. This would, they propose, lead to "unconstrained verb polysemy" because we would have to write different entries for verbs each time we see them appearing in another non-canonical frame. In turn, it also becomes implausible to say that the "caused-motion" meaning is the outcome of the syntactic projections of heads "wipe" and "push", because this suggestion would not be able to express how "sneeze" and "laugh" can be used to describe a caused-motion event as well.

The solution posited by CxG is that neither syntactic nor semantic information is coming from the lexicon. Instead, it is the constructions that determine both semantic interpretation and syntactic representation. Constructions in this sense are semantically interpreted abstract syntactic frames which have their own significance and an existence independent of verbs. For example, the ditransitive construction above, formally represented as "NP V NP in/to NP", is a construction which denotes caused-motion and selects three thematic roles: agent, theme and goal. Crucially, verbs no longer project any syntactic information but only fill in appropriate slots made available by various constructions. Whether a verb can be inserted within a certain construction is determined by the idiosyncratic properties of the verb. Aside from this idiosyncratic information, a lexeme has no further part to play in grammatical theory. While it is true that CxG grammars capture many of the flexibilities displayed by lexemes in a powerful way, they have usually been blamed for being too functionalist, turning the language into a kind of "inventory" of structures, and not appreciating the computational mechanism at all.

So far, we have reviewed how two approaches trying to account for the correspondance between semantics and syntax. While lexicalist theories believe that syntax derives (partially) from semantics and verbal argument structure, constructionist theories believe that none of them are derived from one another. A third view is presented in more compositional approaches like Combinatory Categorical Grammars, e.g. Steedman (1996, 2000) [28][29], which keep the correspondance in a one-to-one development through the derivation. We review this approach below.

### 2.3 Combinatory Categorial Grammar

Combinatory Categorial Grammar (CCG) is a member of the family of Categorial Grammars based on the AB calculus of Adjukiewicz (1935)[30] and Bar-Hillel (1953)[31], whose work centered upon capturing the type structure of natural languages. In the classical AB calculus, grammatical objects are categories which may belong either to an atomic (alternatively referred to as *saturated*, *primitive* or *non-functional*) type or to a function type (alternatively *unsaturated* or *compound type*). Atomic types are denoted with atomic symbols such as $\alpha, \beta, X, Y$ etc. while functions are denoted as $\alpha /\beta_i$ for $i \geq 1$, where $\alpha$ and $\beta$ are categories. A category of type $\alpha /\beta$ is interpreted as denoting a set of expressions which take a category of type $\beta$ as an argument to yield an expression of type $\alpha$ (i.e. the set of functions from $\beta$ to $\alpha$).

Bar-Hillel makes a distinction between functions which take arguments in a forward or backward manner. For example, an expression like $\alpha\,/(\beta)[\gamma]$ denotes a set of functions which combine with an expression of type $\beta$ and an expression type $\gamma$, yielding an expression of type $\alpha$. As noted by McConville (2007: 36)[32], in Categorial Grammar formalisms the same effect is achieved by Schönfinkelization (i.e. currying) of Bar-Hillel's ordered categories with the help of forward and backward slashes /, \: $((\alpha\backslash\beta)\,/\gamma)$. Schönfinkelization turns a function that takes more than one argument into a function that takes those arguments one-by-one in an ordered way. In other words, Schönfinkelization means that any function f that has the form of $i \times \beta \mapsto m$ (from the domain $i \times \beta$ to the range m) is equal to a function g from the domain i to the range $(\beta \mapsto m)$, which is itself function from domain $\beta$ to $m$; i.e. $(i \mapsto (B \mapsto m))$.

CCG assumes a categorical lexicon, and assigns either a function type or atomic type to each lexical item. Consider a toy sentence like *John found the puppy*. The categorical type of each lexical item in this sentence under CCG would be as below:

9. the := NP/N
   John := N
   found := (S\NP)/NP
   puppy := N

There are two functor categories in 9: *the*, which takes an atomic category N to the right to form an NP, and the predicate *find*, which takes an NP to the right to form a VP (=(S\NP)) and another NP to the left to form an S.

In CCG, every syntactic type is paired with a semantic type (i.e. the logical form) which defines the predicate-argument structure of the category in question. The logical form is formalized with typed $\lambda$-calculus. For example, given the lexical-functor category (S\NP)/NP such as *find*, we have the correspondent logical form of this category as in (10):

10. found := (S\NP)/NP : $\lambda x.\lambda y.find'xy$

The $\lambda$-binding on the variables indicate which NPs are interpreted as the object and the subject. The rightmost NP in the syntactic type corresponds to the innermost $\lambda$-bound variable in the semantic type, indicating the variable that the verb applies first, i.e. the object. The predicate-argument structure represented in the CCG logical forms is agnostic to language specific parameters and is assumed to be cross-linguistically universal. For example, one can see that the directionality of application as indicated by slashes (/, \), which belongs purely to the surface syntactic derivation, is not represented in the semantic interpretation. As Steedman (2019)[33] notes, predicate-argument structure is "essentially equivalent to the lexical component of thematic structure in Minimalism, f-structure in LFG, ARG-ST in HPSG, the grammatical function tier of SS".

In CCG, a transparent syntax-semantics interface is maintaned through the derivation of the sentence. Given a set of lexical items with associated syntactic and se-

mantic types, CCG derives the syntactic structure and the semantic intepretation of the sentence compositionally by means of a finite number of combinatory rules[1] and functional application, as in (11).

11.
$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{John}}{\texttt{NP}} \quad \dfrac{\dfrac{\text{found}}{(\texttt{S}\backslash\texttt{NP})/\texttt{NP}} \quad \dfrac{\dfrac{\text{the}}{\texttt{NP}/\texttt{N}} \quad \dfrac{\text{puppy}}{\texttt{N}}}{\texttt{NP}}>}{\texttt{S}\backslash\texttt{NP}}>}{\texttt{S}}<}{}$$

$: john' \quad : \lambda x.\lambda y.found'xy \quad : \lambda x.x \quad : puppy'$

$: puppy'$

$: \lambda y.found'puppy'y$

$: found'puppy'john'$

Important for our purposes is how categorical grammars like CCG represent the argument structure of predicates. Compared to the theories discussed in sections 2.1. and 2.2., it is the one that decorates the lexical-functor categories most richly, by both using a syntactic type defining the subcategorization frame and a semantic type defining the predicate-argument structure of the predicate. Grammatical phenomena which is relevant to predicate-argument structure, such as control or raising, extraction, or pro-drop, are all defined as lexical properties over predicates, that is; as properties of the logical form. For example, subject control verbs such as *promise* have the subject control property defined using $\lambda$-binding in their logical form as in (12).

12.
$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{John}}{\texttt{NP}} \quad \dfrac{\dfrac{\text{promised}}{(\texttt{S}\backslash\texttt{NP})/\texttt{VP})/\texttt{NP}} \quad \dfrac{\text{Mary}}{\texttt{NP}} \quad \dfrac{\dfrac{\text{to}}{\texttt{VP}/\texttt{VP}} \quad \dfrac{\text{win}}{\texttt{VP}}}{\texttt{VP}_{\text{to}}}>}{(\texttt{S}\backslash\texttt{NP})/\texttt{VP})}>}{\texttt{S}}>}{\texttt{S}}<}{}$$

$: john' \quad : \lambda x.\lambda p.\lambda y.promised'(py)xy \quad : mary' \quad : \lambda p.p \quad : \lambda y.win'y$

$\lambda y.win'y$

$: \lambda p.\lambda y.promised'(py)mary'y$

$: \lambda y.promised'(win'y)mary'y$

$: promised'(win'john')mary'john'$

(12) is a case of subject control because *John* is the semantic subject of the infinitival clause *to win*. As can be seen from the semantic type of the verb, this control relation is captured at the level of predicate-argument structure, specifically with the variables *p* and *y*. First, the logical form of *promise* makes it clear that it selects for a clausal complement (specified by variable *p*). Second it also defines that the variable *y*, which

[1] It is beyond the scope of this work to review all the combinatory rules of CCG. The reader is referred to relevant works such as Steedman (2019) [33] and Steedman, (2020) [29] for a detailed overview.

is the subject argument of *promise*, is also the subject of the clause subcategorized for by *promise*.

As another example, consider (13), which is a sentence with a pro-drop subject. In languages which rich inflection like Turkish, agreement morphology on the predicate allows for the subject to be covert in the surface syntax while it is understood at the semantic level.

13. Otobüsü kaçırdı

$$\frac{\frac{\text{Otobüsü}}{\texttt{NP} : bus'} \quad \frac{\text{kaçırdı}}{\texttt{S}\backslash\texttt{NP} : \lambda x.missed'(x, one_{3s})}}{\texttt{S} : missed'\, bus'\, one_{3s}}\ >$$

CCG captures pro-drop in the logical form as well, using an anaphorically bound *one* pronoun as part of the predicate-argument structure representation of the predicate. This ensures that the argument structural properties of the verb *miss* (i.e. that it is a predicate which subcategorizes for two arguments) is accounted for properly in the context of a missing argument in the surface representation.

To summarize, in CCG the logical form has a critical role in determining the predicate argument structure of functor categories. The rich and transparent representation that CCG employs to represent the semantics of predicates not only allows for a transparent syntax-semantics interfaces in all steps of derivation, but it more importantly ensures that argument structural properties of predicates is captured even in the face of data that can be unseen in the surface form (e.g. pro-drop, control etc.).

## 2.4 Conclusion

An important difference between the various approaches to argument structure is what kind of information they assume to exist in the lexicon. In approaches adopted by Fillmore (1968) [21], Gruber [34], Jackendoff (1974) [35], Levin and Rapoport Hovav (2005) [23], among others, lexicon and syntax are two different layers of representation. The valency (or subcategorization) information comes from the lexicon in the form of $\theta$-grids which define the number and semantic type of the subcategorized arguments. Syntax is a CFG determining how various arguments realized in the surface form should be grouped together using phrase structure rules. In these approaches, derivational constraints are defined over phrase structure trees rather than over the lexicon (e.g. using empty categories and/or movement rules).

A similar conceptualization of the lexicon can be observed in CCGs, where lexicon is responsible for specifying for each functor category the type and number of arguments it takes. However, being radically lexicalized, CCG does not assume a separate

layer of representation than the lexicon which would be responsible for syntactic and semantic composition. Each lexical item also has a logical form associated with it which defines the predicate-argument structure of the category in question. Crucially, the logical type of a category differs from its syntactic type in the sense that it accounts for argument structural constraints that the predicate defines over the derivation even when arguments might be missing in the surface representation. As such, derivational constraints in CCG are defined in the lexicon, as properties of the predicate argument structure of a category in its logical form.

As we will review in Chapter 4, the dependency representation, which is the representation we use in the parsing experiments in this thesis, lacks any linguistic information which describes the argument structural properties of predicates. The lack of this kind of information is a major challenge for models which aim to learn grammar from datasets using the dependency representation. As shown in the examples above, linguistic features such as subcategorization or valency explain a predicate's behaviour on unseen data and lack of this information on a predicate strictly constrains learnability of predicate-argument structure to data that the model can see in the particular dataset, which will inevitably be impoverished and sparse. This makes it impossible for the learning model to make good generalizations about verbal behaviour in the face of such data.

# CHAPTER 3

# DEPENDENCY PARSING

In this chapter, we review the two main dependency parsing methods, namely *transition-based* parsing and *graph-based* parsing. As the model that we develop Chapter 5 onwards is also a variety of the graph-based approach, we also review the current state of the art neural implementations of it.

## 3.1 Dependency Parsing

Dependency parsing is the task of resolving the grammatical structure of a sentence by means of constructing a *dependency graph* that represents the syntactic relations which hold in the sentence. The syntactic relations are defined in terms of bi-lexical relationships, or modifier-head relationships, of attachment where each word is connected with the word it modifies, called its head, by means of a dependency arc. The arcs in the graph can also have *labels* which name the grammatical relation that holds between the head and the dependent connected with the arc.

Not all dependency graphs represent linguistically well formed dependency trees. For a dependency graph to be a linguistically well formed *dependency tree*, it should conform to the conditions described in Kübler et al (2009): [36]

1. **Single Rooted**: There should be one and only one *root* node in the graph.
2. **Acyclic**: No node can be the dependent of its own dependent, or one of the dependents of its own dependents. Cyclic paths are not allowed.
3. **Directed**: Each node except for the *root* node has a head.
4. **Spanning**: There exists a path from *root* node to every node.
5. **Connected**: There exists a path connecting every two nodes in the tree.

The **single-rootedness** condition guarantees that the sentence has only one main predicate which all the other constituents (directly or indirectly) modify. The **acyclicity** guarantees that the leaf nodes in the tree are not heads but only modifiers. The **directedness** condition makes sure that every word in the sentence modifies one other

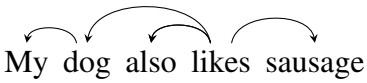


Figure 1: Dependency tree of sentence *'My dog also likes sausage'*

word. Combined with **acyclicity**, it also guarantees that words cannot be modifiers of themselves.

Dependency trees are linear representations, however, from Figures 1 and 2, it is easy to see that they capture the hierarchical structure of the sentence to a certain extent. A significant difference is that compared to a phrase structure tree, we see from Figure 2 that in dependency representation there is no intermediate layer which represents phrasal constituents.

Besides the aforementioned well-formedness conditions, some researchers have also defined a *projectivity* property for dependency trees which disallows arcs crossing each other in a dependency graph. More formally, this property can be defined as follows: a *projective* dependency arc is one where there is path from the head word to every other word situated in between the head and the dependent positions connected with the arc. Given this, a projective dependency tree is defined as one where all the dependency arcs in the tree is projective.

It can be understood from the definition of projectivity above that it is mainly related to word order. In the parsing literature, a lot of languages have been noted where the word order in grammatical sentences does not satisfy projectivity. This is specifically true in flexible word order languages such as Czech, Hindi and Turkish (Kuhlmann and Nivre, 2006) [37], (Collins et al., 1999) [38]. Therefore, projectivity is not defined as a constraint on the set of possible dependency trees, but is regarded only as a property that distinguishes certain types of trees from others.

A *dependency parser* is a model that generates dependency trees. A parser can be grammar-driven in which case it depends on a set of formal grammar rules that deter-

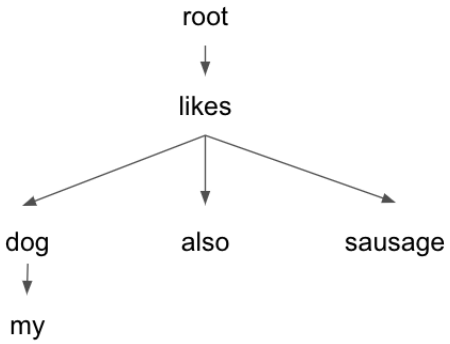


Figure 2: Hierarchical structure of sentence '*My dog also likes sausage'*

mine whether a dependency tree is part of a formal language. A data-driven parser, on the other hand, is one led by a machine learning algorithm which induces a parsing model that constructs well formed dependency trees from annotated dependency treebanks. It is these type of parsers we will be dealing with in this thesis.

A data-driven dependency parser can be analyzed from two points of view:

1. **Learning Method**: The ML method that is used to induce the parsing model from data.
2. **Parsing Algorithm**: The algorithm that generates the dependency tree given a sentence and a learned parsing model.

In terms the learning method, most recent approaches have been using some variety of Deep Neural Networks (DNN). However, traditional ML models such as SVMs (Yamada and Matsumoto, 2003) [39] or Perceptrons (Collins, 2002) [40] have also been widely used before the recent advancement on DNNs.

In terms of the algorithm being used, two well established methods represent current state of the art: the *Transition Based Parsing* method and the *Graph Based Parsing* method (McDonald and Nivre (2007)[41]). The distinction between the two methods lies in how they conceptualize the parsing problem.

### 3.1.1 Transition Based Parsing

For a transition-parser, parsing is a problem of sequential decision making. Transition parsers employ a state machine that processes the sentence left-to-right (token by token) and builds the dependency tree in a bottom-up fashion.

At the heart of the transition parser is the notion of a Configuration (C), which represents the state of the dependency tree at any given timestep as the parser incrementally builds up the tree. Formally, a configuration is a data structure that consists of three components:

- A Stack ($\delta$): Holding the words that are (partially) processed.
- A Buffer ($\beta$): Holding the words that are yet to be processed.
- A set of dependency arcs ($A$): A list of tuples $(w_i, r, w_j)$ keeping track of words $(w_i, w_j)$ that are connected by a dependency arc *r*.

During parsing, the transition parser modifies the configuration based on the transition decisions it takes. In the initial configuration, i.e. before the parser starts processing the sentence, the buffer holds all the words and the stack is empty (except for the dummy token representing $root$ of the tree), i.e. C: $\delta = [root], \beta =$

$[w_1, w_2, ..., w_n], A = ()$. As the parser processes the sentence, it takes transition decisions that modify the configuration. In each transition, words might be pushed to the stack, removed from the stack, or removed from the buffer. The set of dependency arcs $A$ is updated accordingly to keep track of the head-dependent relations identified during this parsing process.

As can be understood from the brief overview, in a transition parser the model parameterized over a set of actions. The actions help the parser move from one state to another. At its basic, arc-standard version, the parser chooses one of three transition actions: LEFT-ARC, RIGHT-ARC, and SHIFT. Their formal definitions are given below based on (Nivre, 2008)[42]

*Transitions for Arc-Standard Transition Parsing*

- `TRANSITION LEFT-ARC`: $(\delta|w_i,\ w_j|\beta,\ A) \rightarrow (\delta,\ w_j|\beta),\ A \cup \{w_j, r, w_i\}$

  Given a word $w_i$ at the top of the stack and a word $w_j$ in the buffer, assigns j as the head of i. Pops i from the stack as it's head is now found.

- `TRANSITION RIGHT-ARC`: $(\delta|w_i,\ w_j|\beta,\ A) \rightarrow (\delta,\ w_i|\beta),\ A \cup \{w_i, r, w_j\}$

  Given a word $w_i$ in the stack and a word $w_j$ in the buffer, assigns $w_i$ as the head of $w_j$. Removes $w_j$ from the buffer and puts $w_i$ to the buffer instead. This last step, i.e. replacement of $w_j$ with $w_i$ in the buffer, is done so that the parser can determine at a later stage whether $w_i$ might have a head to its left in the stack.

- `TRANSITION SHIFT`: $(\delta,\ w_j|\beta,\ A) \rightarrow (\delta|w_j, \beta,\ A)$

  Pushes the word at the top of the buffer to the stack. This is the transition applied when the parser cannot find a head-dependent relation between the word in the stack and the buffer (left-arc or right-arc does not apply). The word in the buffer is pushed to the stack for later processing as the sentence unfolds.

Table 1 below traces an example parse of the sentence *The cat sat on the mat* based on the transitions defined above.

Given a configuration, the task for an ML classifier is to classify the correct transition action to take at each step. At training and inference time, the classifier uses the linguistic properties of the words on the stack and the buffer as features.

Two main important properties of transition based parsers is that they parse sentences in linear time and they guarantee generating projective trees. Projectivity is guaranteed due to the fact that the tree is built in a bottom up fashion. Linearity is guaranteed because the system has no backtracking and each {x,y} token pair is evaluated at most once. Notably, this latter property also makes transition parsers prone to error propagation. In other words, if the parser has made a wrong attachment decision between some early token pairs in the sentence, this decision might lead to more errors in the later stages of parsing and there is no way to correct this decision later based on more signals.

| TRANSITION | STACK | BUFFER | RELATIONS |
|---|---|---|---|
| | [*root*] | [The, cat, sat, on, the, mat] | $\varnothing$ |
| SHIFT | [*root*, The] | [cat, sat, on, the, mat] | $\varnothing$ |
| LEFT-ARC | [*root*] | [cat, sat, on, the, mat] | $A = \{cat, DET, the\}$ |
| SHIFT | [*root*,cat] | [sat, on, the, mat] | $A = A$ |
| LEFT-ARC | [*root*] | [sat, on, the, mat] | $A = A \cup \{sat, NSBJ, cat\}$ |
| SHIFT | [*root*, sat] | [on, the, mat] | $A = A$ |
| SHIFT | [*root*, sat, on] | [the, mat] | $A = A$ |
| SHIFT | [*root*, sat, on, the] | [mat] | $A = A$ |
| LEFT-ARC | [*root*, sat, on] | [mat] | $A = A \cup \{mat, DET, the\}$ |
| RIGHT-ARC | [*root*, sat] | [on] | $A = A \cup \{on, POBJ, mat\}$ |
| RIGHT-ARC | [*root*] | [sat] | $A = A \cup \{sat, PREP, on\}$ |
| RIGHT-ARC | [] | [root] | $A = A \cup \{root, ROOT, sat\}$ |
| SHIFT | [*root*] | [] | $A = A$ |

Table 1: Arc-standard parse of sentence '*The cat sat on the mat*'

In the literature, certain modifications to the arc-standard system have been proposed to overcome some of the challenges that such systems face. One such modification is adding a REDUCE transition to the set of actions which can remove words from stack any time (i.e. not only when they are a left dependent) as long as they have a head (Nivre, 2003, 2004)[43] [44]. This change also allows for modifying the behaviour of RIGHT-ARC transition to let right-dependents attach their left heads right away, instead of only after all their dependents have been identified. The resulting system is called *arc-eager*, for which the transition actions are defined below.

*Transitions for Arc-Eager Transition Parsing*

- `TRANSITION LEFT-ARC`: $(\delta|w_i,\ w_j|\beta,\ A) \rightarrow (\delta,\ w_j|\beta), A \cup \{w_j, r, w_i\}$
- `TRANSITION RIGHT-ARC`: $(\delta|w_i,\ w_j|\beta,\ A) \rightarrow (\delta|w_i, wj,\ \beta), A \cup \{w_i, r, w_j\}$
- `TRANSITION SHIFT`: $(\delta,\ w_j|\beta,\ A) \rightarrow (\delta|w_j,\ \beta,\ A)$
- `TRANSITION REDUCE`: $(\delta|w_i,\ \beta, A) \rightarrow (\delta,\ \beta,\ A)$

Table 2 below shows the arc-eager parse of the sentence *The cat sat on the mat*.

There have been other proposals in the literature to modify the transition system in one way or another. It is beyond the scope of this work to review them all. However, some notable ones worth mentioning are Nivre (2009) [45] adding a SWAP transition to the system to deal with non-projectivity and Qi and Manning (2007)[46] defining an *arc-swift* transition system that allows tokens farther apart from one another to attach rather than just the immediately adjacent ones in the stack and the buffer.

Transition parsers can easily be extended to become joint systems that take parsing and labeling decisions together. The method is to expand the transition types to create a labeled version of them for each of the available dependency labels (i.e. LEFT-ARC -> LEFT-ARC$_{NSUBJ}$, RIGHT-ARC -> RIGHT-ARC$_{OBJ}$ and so on). In this labeled

| TRANSITION | STACK | BUFFER | RELATIONS |
|---|---|---|---|
| | [*root*] | [The, cat, sat, on, the, mat] | $\varnothing$ |
| SHIFT | [*root*, The] | [cat, sat, on, the, mat] | $\varnothing$ |
| LEFT-ARC | [*root*] | [cat, sat, on, the, mat] | $A = \{cat, DET, the\}$ |
| SHIFT | [*root*,cat] | [sat, on, the, mat] | $A = A$ |
| LEFT-ARC | [*root*] | [sat, on, the, mat] | $A = A \cup \{sat, NSUBJ, cat\}$ |
| RIGHT-ARC | [*root*, sat] | [on, the, mat] | $A = A \cup \{root, ROOT, sat\}$ |
| RIGHT-ARC | [*root*, sat, on] | [the, mat] | $A = A \cup \{sat, PREP, on\}$ |
| SHIFT | [*root*, sat, on, the] | [mat] | $A = A$ |
| LEFT-ARC | [*root*, sat, on] | [mat] | $A = A \cup \{mat, DET, the\}$ |
| RIGHT-ARC | [*root*, sat, on, mat] | [] | $A = A \cup \{on, POBJ, mat\}$ |
| REDUCE | [*root*, sat, on] | [] | $A = A$ |
| REDUCE | [*root*, sat] | [] | $A = A$ |
| REDUCE | [*root*] | [] | $A = A$ |

Table 2: Arc-eager parse of sentence '*The cat sat on the mat*'

parsing scenario, classifiers are trained to learn *typed/labeled transitions* and predict such transitions at inference time.

### 3.1.2 Graph Based Parsing

Graph-based parsing defines the dependency parsing as a problem of finding the maximum spanning tree (MST) amongst the set of all possible trees in a dependency forest. The most prominent approach in this line of research is the algorithm developed in studies like McDonald et al. (2005) [47], McDonald and Perreira (2006) [48], and related work.

MST parsing is *arc-factored* in the sense that it parameterizes the parsing model over the set of all possible dependency arcs. The task is to determine, among the exhaustive list of possible arcs connecting any two nodes in any sentence, weights for the optimal set of arcs that span over all the nodes and preserve the well-formedness conditions for dependency trees defined in section 3.1. Let us explore how this is done in detail.

Given a sentence S such that $S = w_1, w_2, ...w_n$ where $w_1, w_2, ...w_n$ is the set of words (or *nodes* in graph-theoretic terminology), the set of all possible dependency arcs *R* for the dependency tree *T* of *S* can be defined as in equation (1):

$$\lambda_{(T)} = \lambda_{(w_i,r,w_j)} \epsilon\ R, for\ each (w_i, r, w_j) \epsilon\ T \tag{1}$$

To give an example, consider a sentence like *Ada enjoyed the book*. $\lambda_T$ for this sentence would correspond to the graph on the left in Figure 3, representing the set of all possible arcs for all the nodes in S. An MST parser's job is to score all the arcs connecting any two nodes in this graph based on a learned scoring model and then for each arc $(w_i, r, w_j) \epsilon\ T$, keep the one that has the highest score in the ultimate

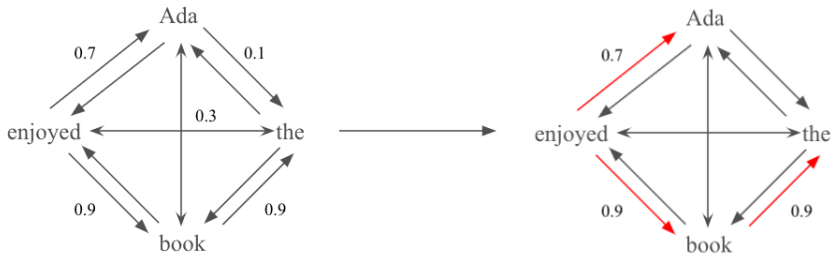


Figure 3: Maximum spanning tree representation of the sentence '*Ada enjoyed the book*

dependency tree. To put it in another way, it aims to retrieve the red arcs on the right from the one on the left based on a scoring function.

The scoring function is an ML model trained to assign scores to all the possible arcs in the tree. As such, the score of the ultimately constructed dependency tree is a function of the scores of the individual arcs such that:

$$score(T) = argmax\ \Sigma_{(w_i,\ r,\ w_j)\epsilon A}\ \lambda_{(T)} \tag{2}$$

Based on the discussion above, it is easy to see that for an MST parser the parsing problem reduces to finding the dependency tree whose arc scores sum to the maximum value. Given that arcs attach dependents to their heads, another way to describe this is that for an MST parser the parsing problem is finding the most probable head for each word in the tree. As a result, we can say that for graph based methods dependency parsing is a *head selection* problem.

Differently from the transition-parsers, MST parsers do not guarantee a well-formed tree. This is for two reasons. First, they do not guarantee that the maximum spanning tree will conform to the **single-rootedness** condition. For example, it might happen that the ML model scores two words as dependents to the ROOT node, meaning it selects two ROOTs for the sentence. Second, MST parsers do not guarantee **acyclicity**. This is due to the fact that there is no constraint on the arc-factored scoring process to assign highest scores only to non-cyclic paths. Such problems are resolved by post-processing the generated trees using the Chu-Liu-Edmonds (CLE)[49][50] algorithm which breaks any cyclic paths and ensures that only one ROOT is maintained in the tree. Without going into too much detail, the CLE algorithm is a recursive algorithm that breaks cycles in a tree with a pruning procedure which collapses nodes included in the cycle into a new single node and recalculating the arc weights in the tree. The process is guaranteed to find the maximum spanning tree without any cycles, but due to the recursive calls leads to a time complexity of $O(n^3)$[1].

[1] For a detailed explanation of the CLE algorithm with examples, the reader is referred to Kübler et al (2009, chapter 4)[36]

### 3.2 Neural graph based parser implementations

With the recent advancements in DNNs, a variety of neural MST parser implementations have appeared in the dependency parsing literature. In this section, we briefly talk about some of them, and mainly discuss the currently accepted state of the art MST parser model, i.e. Stanford's Biaffine Parser[51][52][53], in detail.

Kipperwasser and Goldberg [54] presents the first neural implementation of MST parsing, where they implement birectional LSTMs to encode input tokens, on top of which a multi layered perceptron (MLP) predicts dependency arcs between tokens. The architecture proposed in this work was extended in Stanford's Biaffine Parser where the outputs from the LSTM encodings are used in two biaffine classifiers where one computes the head *y* for each token *x* and the other computes the dependency label *l* for each dependency arc (*x, y*).

In the Stanford Parser, the input to the model is a sequence of tokens and PoS tags. The model uses multilayered bidirectional LSTMs to encode the features of each token-PoS pair, and then uses the output hidden state of the last LSTM layer to create four different vector representations for each input token. These vector representations are created with the help of 4 different MLPs with ReLU activation, each of which transforms the hidden state output into a different vector as in equations 3-6 (Dozat et al, 2017: 21):

$$h_i^{(arc-dep)} = MLP^{(arc-dep)}(h_i) \tag{3}$$

$$h_i^{(arc-head)} = MLP^{(arc-head)}(h_i) \tag{4}$$

$$h_i^{(rel-dep)} = MLP^{(rel-dep)}(h_i) \tag{5}$$

$$h_i^{(rel-head)} = MLP^{(rel-head)}(h_i) \tag{6}$$

Each vector in equations 3-6 creates a different representation of input token *t*. Equation 3 creates a representation of *t* as a dependent seeking its head, equation 4 as a head seeking its dependents, equation 5 creates a representation of *t* as a dependent seeking its dependency label, and equation 6 as a head word determining the dependency labels of its dependents. This is graphically represented in Figure 4.

Crucial in the Stanford Parser is the two biaffine classifiers that apply to the outputs of the MLP transformations in 3-6. The first biaffine classifier, shown in equations 7-8 below, applies to the head and dependent vector representations in 3-4 and outputs an *arc* prediction ($y_i^{(arc)}$) for each token *i*. In other words, this classifier selects the most likely head for token *i* by outputting for every token $y_i$ the probability of its being the head of token *i*. In turn, the second biaffine classifier applies to the *rel*

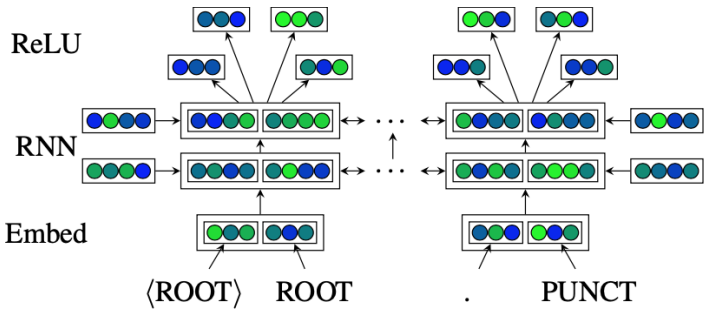


Figure 4: Stanford Parser Architecture

vectors in equations 5-6 and given a head-dependent pair, outputs the probability of a dependency label *rel* for this pair, following equations 9-10 (Dozat et al, 2017:22)[2].

$$s_i^{(arc)} = h^{(arc-head)} W^{(arc)} h_i^{(arc-dep)} + h^{(arc-dep)} b^{T(arc)} \tag{7}$$

$$y_i'^{(arc)} = argmax_j s_{ij}^{(arc)} \tag{8}$$

$$s_i^{(rel)} = h_{y_i'^{(arc)}}^{T(rel-head)} \cup^{(rel)} h_i^{(rel-dep)} + W^{(rel)}(h_i^{(rel-dep)} \oplus h_{y_i'^{(arc)}}^{(rel-head)}) + b^{(rel)} \tag{9}$$

$$y_i'^{(rel)} = argmax_j s_{ij}^{(rel)} \tag{10}$$

Models that are similar to Stanford Parser have been proposed also in Hashimoto et al. [55] and Zhang et al. [56]. Hashimoto et al. present a multi-task model which jointly does PoS tagging, chunking and dependency parsing at the same time in a similar LSTM based framework. Zhang et al. also use a similar model where the difference is that in the decoding stage they retrieve the best tree not using the CLE algorithm but by a parse reranker. The Stanford Parser has won the CoNLL shared task in 2017, improving the state of the art parsing performance across a variety of languages and datasets. In Chapter 5, we run our experiments on the same CoNLL datasets as the Stanford Parser and compare our results to this parser.

[2] In all the equations in 7-10, *W* stands for the weight vectors, *h* for the hidden state outputs from the LSTM layers, and *b* for the bias terms.

## 3.3 Conclusion

In this chapter, we have reviewed the transition-based and graph-based dependency parsing algorithms and discussed how the conceptualize the parsing problem. While for transition-based parsers, dependency parsing is a problem of sequential decision making using a state-machine, graph-based parsers interpret the parsing problem as a problem of head-selection using graph processing algorithms.

As we conclude this section, let us clarify that the models that develop in this thesis falls together with the graph-based approach. However, it differs critically from the state of the art implementations of this approach in terms of how it generates a labeled dependency tree. As can be understood from our review above, both the Stanford Parser, and its variations cited in section 3.2 set up the dependency parsing task in a *head-first* manner. In other words, first the dependency arcs are generated using token features, then for each dependency arc $(x,y)$, a dependency label is predicted. Contrary to this, the models that we develop in this thesis are *label-first* as we try to account for the predicate and argument roles in the sentence by predicting dependency labels first, and then using these predictions as inputs generating the dependency tree. As we show in our multilingual experiments in Chapter 5, accurate prediction of these labels early in the parsing stage improves parsing performance in a variety of languages (but not in all of them).

# CHAPTER 4

# THE DATA

## 4.1 Introduction

In this chapter, we review the properties of the data that will form the basis of our experiments in the following chapters. Mainly, we review properties of UD dependency treebanks, discussing, when relevant, its limitations in capturing important linguistic distinctions especially with respect to predicate argument structure. Due to the parsing strategy proposed in this thesis, we will also focus in more detail on the set of dependency labels used in UD treebanks.

## 4.2 Dependency Structure and Grammatical Relations in UD

The UD effort aims to develop a unified framework for representing syntactic structure across a large number of linguistically diverse languages. This is achieved by adopting a *dependency grammar* formalism which, as we have reviewed in the previous chapter, represent syntax using binary asymmetrical links between words that modify one another.

There are several design principles of UD which are discussed in detail in the main UD paper[7]. An important principle is *simplicity*. UD authors state that the goal is to develop treebanks in such a way that linguistic structures can be easily used and understood by non-linguists and can be rapidly and consistently annotated by annotators. This goal forces UD to be as theory-agnostic as possible, avoiding constructs or distinctions that are particular to a specific grammatical theory (e.g. empty categories in *Transformational Grammars*). On the other hand, it can sometimes lead to oversimplification, or a failure to capture, some grammatically relevant distinctions in the linguistic representation.

Another primary goal of UD is *universality*, i.e. developing a representation that can capture the linguistic properties of languages with different typologies. This is mainly achieved by employing a universal set of tags representing grammatical relations (i.e. dependency labels). Even though the core set of dependency labels are universal and unified across languages, UD allows them to be further extended to account for a predominant linguistic phenomenon which could be observed in some language or

language family. For example, the *dislocated* tag is mainly used to mark nominals which introduce the *Topic* of discussion without being part of the predicate-argument structure of the sentence in East Asian languages like Japanese. Maintaining a unified representational framework while trying to capture as many languages with different linguistic properties as possible is challenging and can sometimes cause semantic overloading of particular dependency labels. For example, in many treebanks for languages which do not have the same properties as Japanese, the *dislocated* tag is used to represent any kind of nominals extracted from their typical positions, including those that are in fact part of the predicate-argument structure of the sentence.

The relatively simple representation that UD provides makes it suitable for parsing with high accuracy, and its universal approach makes it scalable to various languages. However, the commitment to simplicity and universality does come with certain linguistic drawbacks.

### 4.2.1 Argument-Adjunct Distinction

The binary representation of syntactic relations in UD (or in dependency grammars in general) leads to a significantly simpler syntactic tree compared to e.g. phrase structure trees. As a result, it is impossible to account for certain linguistic distinctions concerning argument vs. adjuncts within this representation. Consider examples in (1) below.

1. a. John is a lazy student incapable of success.
   b. John is a student incapable of success.
   c. *John is a student lazy incapable of success.

The constraint on word order exemplified in a sentence (1c) is explained in theories that assume a phrase structure such as the X' Theory of Jackendoff 1977 [57] by stipulating word level and phrase level adjunction rules as in (2):

2. a. N -> AP N
   b. NP -> NP AP

According to this rule, for English, word level adjuncts adjoin their heads in a head-final fashion while phrase level adjuncts in a head-first fashion. This is represented in phrase structure grammars by an intermediate layer of representation (e.g. X') that groups together heads and their adjuncts, which is a different layer than the one that groups together heads and their complements (e.g. XP). In contrast, when the UD representation of the same sentence is considered (Figure 5), there is no difference between the grammatical status of the modifiers *a, lazy* and *student* with respect to their head *student*. All the modifiers, whether determiners, adjectives, or phrasal heads, modify the NP head *student* in the same way.

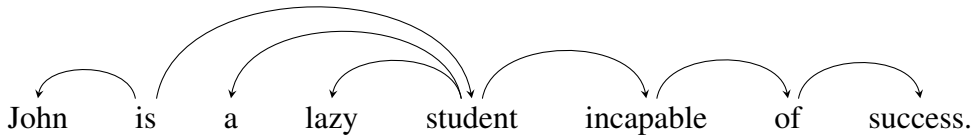


Figure 5: Dependency tree of sentence *'John is a lazy student incapable of success'*

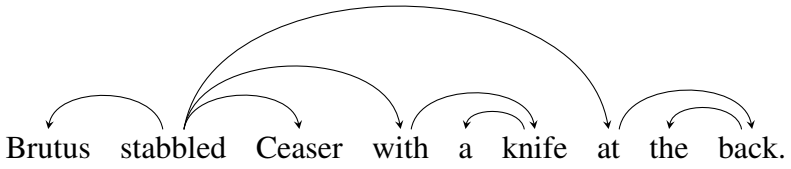


Figure 6: Dependency tree of sentence *'Brutus stabbled Ceaser with a knife at the back'*

The same situation applies to verbal complementation where dependency formalism does not distinguish, at least in the dependency tree, between complements that are required by the verb (arguments) vs. modifiers that are optional for the verb (adjuncts). This means that UD dependency tree representation is agnostic to linguistic notions such as subcategorization and/or valency.

In Figure 6, the adjuncts *(with a knife)* and *(at the back)* are linearly attached to verbal root *stabbed*. In this representation, they do not have a more significant relationship with the verb than arguments that are actually subcategorized by the verb: *Brutus* and *Ceaser*.

The relatively impoverished syntactic representation that dependency formalism uses leads UD to define a rich set of dependency labels to do the grammatical heavy-lifting that cannot be represented otherwise in the dependency trees. The dependency labels in UD mark the grammatical function of each word in the sentence. In terms of marking distinctions relevant to predicate-argument structure, two sets of dependency labels are defined: those that mark core modifiers (i.e. arguments) and those that mark non-core modifiers (i.e. adjuncts), as in Table 3.

Table 3: Core vs. non-core arguments in UD

| | Subject | Object | Non-core |
|---|---|---|---|
| **nominal** | nsubj | obj/iobj | obl |
| **clausal** | csubj | ccomp | advcl |
| **open** | | xcomp | |

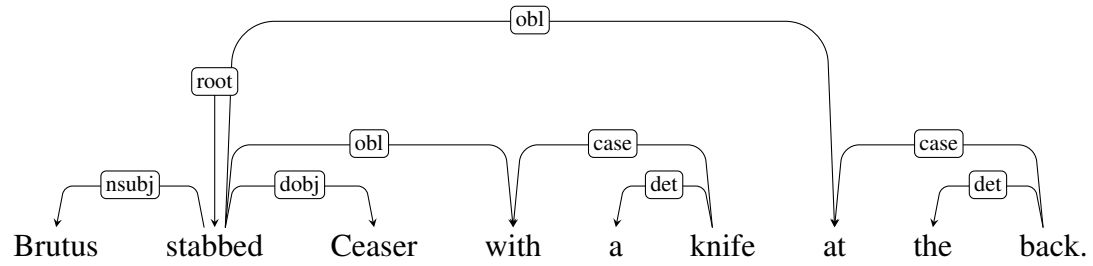


Figure 7: Dependency tree of sentence *'Brutus stabbed Ceaser with a knife at the back with dependency labels.'*

The distinction between *nsubj* and *csubj* is whether the subject is a NP or a clause, similarly for *ccomp* vs NP complements such as *obj/iobj*. Using these labels, the sentence in Figure 6 would then be annotated as Figure 7.

There are two important points to mention about how UD handles adjuncts. First, all adverbial clause modifiers of a verb get an adjunct interpretation in UD. Second, the *obl* label is used as an umbrella term to cover all prepositional phrase (PP) modifiers of the verb (see Figure 7). That is, UD treats all prepositional phrase modifiers of verbs as adjuncts. This is an important problematic linguistic aspect of UD in terms of capturing important distinctions in predicate-argument structure of different verbs. To give some examples, this treatment does not capture the semantics of *verbs of perception* like *look, stare, peep* (cf. Levin (1993: 187) [22] which select for a PP complement.

3. a. I stared [at the restaurant]$_{obl}$.
   b. I fell [at the restaurant]$_{obl}$.
4. a. *I stared
   b. I fell.

Similarly, the use of *obl* label misses the fact that in *dative alternation*, the PP variant is actually a complement of the verb rather than its adjunct.

5. a. I gave [John]$_{dobj}$ [the book]$_{iobj}$.
   b. I gave [the book]$_{dobj}$ [to John]$_{obl}$
   c. I gave the book [to John]$_{obl}$ [at the restaurant]$_{obl}$.
   d. *I gave the book [to John]$_{obl}$ and [at the restaurant]$_{obl}$.

The constraint on conjunction in example (5d) indicates that [at the restaurant] has a different status than [to John] in the context of the verb *give*. In phrase structure grammars, this would be represented by the adjunction rules as explained above. In the current dependency annotation scheme as being followed by UD, the overloading

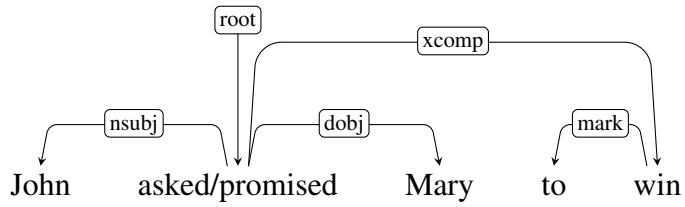


Figure 8: Dependency tree of sentence *'John asked/promised Mary to win'*.

of the dependency label *obl* to cover all kinds of PP complementation fails to capture lexical properties of different verbs.

The problematic status of the argument-adjunct distinction (AAD) in UD can be enumerated with more examples as in Przepiorkiwski and Patejuk (2018) [58]. A fair summary would be to state that while UD manages to differentiate adverbial adjuncts with the dependency label *advcl* successfully, due to its overuse of dependency label *obl*, it fails to capture distinctions relevant to argument structure in PP complementation.

### 4.2.2 Control and Raising

UD borrows the *xcomp* label in Table 3 from Lexical-Functional Grammars[59][60] and uses it for marking *open clausal complements* or, as stated in the UD guidelines, complements without their own subject. These are typically control clauses or raising structures, where the semantic subject of a clausal complement is controlled either by the subject or the object of the matrix clause or receives exceptional case marking (ECM) from the matrix verb.

6. *Subject Control*
   a. John *yearns* to win.
   b. John *promised* Mary to win.
7. *Object Control*
   a. John *asked* Mary to win .

In the UD representation of sentences 6-7, the matrix verbs *(yearn, promise)* would be connected to the head of the clausal complement *win* with the *xcomp* dependency label. Therefore, the dependency structure of sentences like (6b) and (7a) would be exactly the same, as shown in Figure 8. One can see that this representation mistreats the lexical properties of verbs like *ask* vs *promise*, as it fails to represent that one is subject control while the other is object control. One solution to this problem would be to implement an additional dependency arc between *John* and *win* in the subject control cases, and *Mary* and *win* in the object control cases. However,

well-formedness constraints for dependency trees disallow such a treatment because dependency grammars require every word to have only one head.

The aforementioned problem also applies to raising structures, which are equally problematic for dependency grammars. Differently from control predicates, raising predicates such as *seem, appear, want* do not semantically select the argument that appears as their subjects or objects in the surface representation. This can be understood from 8 and 9 below.

8. *Raising to Object*
   a. The teacher *wanted* us to read the whole chapter.
   b. *We were *wanted* by the teacher to read the whole chapter.
9. *Object Control*
   a. The teacher *asked* us to read the whole chapter.
   b. We were *asked* by the teacher to read the whole chapter.

The passivizability test shows that, in contrast to the verb *ask* in (9a), the syntactic object *us* is not the semantic object of the verb *want* in (8a), which makes passivization ungrammatical in (8b). This is a subtle distinction in predicate-argument structure of the two predicates deriving from their lexical semantic properties, about which dependency grammars have nothing to say. In the dependency representation, the word *us* would be marked as the direct object of the matrix verbs in both raising-to-object and object-control cases. UD has no other mechanism or a representational layer which accounts for the fact that in one of the cases the surface object is not subcategorized by the matrix verb.

### 4.2.3 Nominal Modification

Nominal complementation in UD is less problematic than verbal complementation. The basic UD inventory consists of 8 nominal dependents, which are listed in Table 4. Many of the labels are self explanatory and do not require further explanation. For some of the less clear ones: the *appos* label stands for *appositional modifier* structures, i.e. when two noun phrases which refer to the same entity are used side by side in a sentence: "*John, my brother, is 25 today*". UD uses the *acl* tag to mark relative clause modifiers of nominal heads.

Among the set of nominal modifiers, *case* is another label besides *acl* that has clausal significance. The *case* tag is introduced relatively recently to UD to introduce adjuncts headed by adpositional categories, as in Figure 9. UD states that this treatment achieves higher parallelism between prepositional phrases and subordinate clauses which are introduced by complementizers (such as "that" in English). However, besides the fact that it is not clear why such a parallelism is needed, this treatment quickly appears problematic for a language like Turkish which is head-final and where

Table 4: Nominal modifiers in UD

| | | | |
|---|---|---|---|
| **nominal** | nmod | appos | nummod |
| **clausal** | acl | | |
| **adjectival** | amod | | |
| **function words** | det | clf | case |

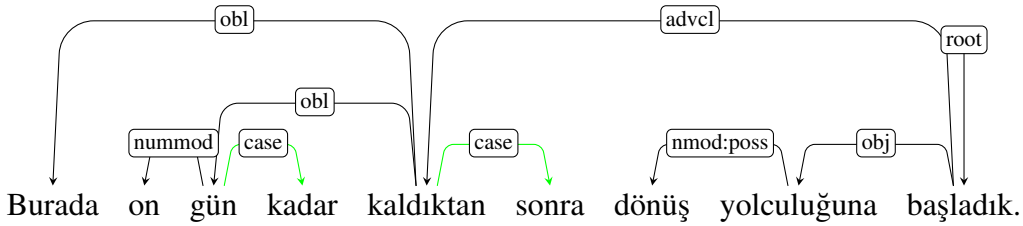


Figure 9: Dependency tree of sentence *'Burada on gün kadar kaldıktan sonra dönüş yolculuğuna başladık.'*

dependencies are normally right-to-left except in post-verbal constructions. As can be seen in Figure 9, the use of *case* dependency label introduces rightward dependencies to the structure, going against the directionality of the language.

A more important problem with the use of *case* label is that it does not capture the fact that adpositions are categories with an internal argument structure (Hale & Keyser, 2002)[61], projecting a complement position in the syntax.

10. a. I left after [the meeting]$_{NP}$.
    b. I left.

11. a. [Okuldan]$_{NP}$ sonra eve gitti.
    b. Eve gitti

In both 10 and 11, the paranthesized NPs are required by the relative adpositions rather than the verbs and therefore should be headed by the adpositions but not the verbs. The current UD scheme, marking these phrases as heads of adpositions, is contrary to this linguistic reality, which leads to further problems in terms of assigning the correct predicate-argument structure to certain sentences. For example, *verbs of putting* subcategorize for a LOCATION argument headed by an adposition (Croft, 1991 [62]).

12. a. I put the books on the shelf.
    b. *I put the books.

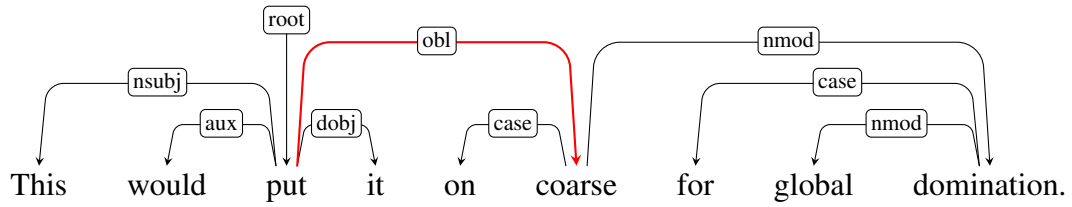


Figure 10: Dependency tree of sentence *'This would put it on coarse for global domination.'*

The current UD representation, with the aforementioned treatment of adpositional phrases, has no option but to treat these phrases as adjuncts of the verb (using the *obl* label), rather than as arguments of the preposition *on*, as in Figure 10 (which is a sentence in the English Web Treebank).

### 4.3 Conclusion

Learning predicate-argument structure means learning the lexical-properties of the predicates: what kind of arguments they select, how many arguments they subcategorize for, how do they establish grammatical relations with the arguments they do and do not select. We have seen that certain properties of UD makes it impossible to learn some of these distinctions using a dependency framework. First, the semantic overloading of the *obl* label fails to distinguish between verbs that require PP complements from the ones that have PP adjuncts. Second, the *xcomp* label does not capture the nature of the grammatical relation between the verbs and their objects or subjects in raising vs. control structures. Finally, the *case* label does not account for the fact that adpositions have their own selectional properties with an internal argument structure.

Besides the points mentioned above, a more important problem with the UD representation in terms of its suitability for learning predicate-argument structure is that it does not have adequate means to represent the restrictions that the predicates impose on syntax as part of their lexical semantic properties. As we have reviewed in Chapter 2, transformational grammars mark such restrictions by means of $\theta$-grids accompanied by argument realization rules. More compositional approaches like CCGs, on the other hand, handle them by means of a logical form which transparently encode such restrictions over the predicates at the level of semantic representation. In UD, however, there is no explicit marking of semantic notions such as subcategorization or valency defined over lexical items which have internal argument structure. As we will see in chapter 5, when such restrictions are not explicitly marked over predicates, there is no way for a dependency parser to induce them from dependency trees well enough. This is mainly for two reasons. The first reason is the fact that certain distinctions in predicate argument structure only apply at the level of semantics and will never be visible over the surface structure of the sentence unless explicitly marked in some way in the linguistic representation (over lexical items, logical forms, syntactic

trees etc). The second reason is the problem of data sparsity. In lack of an explicit representation of argument structural properties over predicates, a parser can only induce verbal behaviour from a dependency treebank if the treebank is rich enough to expose all possible predicate-argument combinations that a verb can appear in. However, no treebank will be as rich, and some treebanks will be more sparse due to the linguistic properties of the specific language which might allow arguments to be more covert in the surface structure. Faced with sparseness, the generalizations that the parser makes will be biased towards the structures that it overtly sees in the surface representation because it will not have any priors about the structures that it does not see.

# CHAPTER 5

# LABEL FIRST PARSING

In this chapter, we report the results of our experiments with label first dependency parsing: a dependency parsing strategy that first predicts dependency labels for each word before building the dependency tree for the sentence. We show that accurate prediction of dependency labels at the word level before dependency arcs improves parsing accuracy for unlabeled attachment score (UAS) and labeled attachment score (LAS) metrics. As was discussed in the previous chapters, in UD most of the dependency labels encode the grammatical function (or grammatical role) that a word plays as part of the overall predicate argument structure of a sentence. The hypothesis is that provided that these grammatical functions are accurately predicted, a parser that has early access to predicate and argument roles at inference time will more accurately figure out the syntactic relationships (i.e. dependency arcs) in the sentence. We test the developed model in multiple languages and report significant improvements on both UAS and LAS on a few of them.

Another aim of the chapter is to analyze why in certain languages improvements are more significantly observed than others. We specifically analyze the errors of the model with respect to accurate identification of predicate-argument structure and argue that dependency formalism is more suitable for inducing argument structural information for certain type of languages than others.

## 5.1 Introduction

As was discussed in Chapter 3, approaches to dependency parsing can be divided into two as transition-based and graph-based [41].

For a transition-parser[43][42], parsing is a problem of sequential decision making. Transition parsers employ a state machine that processes the sentence left-to-right and builds the dependency tree in a bottom-up fashion. At the heart of transition parser is the notion of a Configuration (C) which represents the state of the dependency tree at any given timestep. Transition parsers parameterize the parsing model over a set of shift-reduce decisions. These actions, or transitions, modify the Configuration as they identify head-modifier relations (dependency arcs) and build a dependency tree in a bottom-up fashion during the processing of the sentence.

On the other hand, graph-based systems solve the dependency parsing problem by factoring out the sentence into a unidirected, acyclic dependency graph using graph algorithms. This can be achieved by employing the Maximum Spanning Tree (MST) algorithm which tries to find the highest scoring tree for a sentence amongst the set of all possible trees[47][48]. These systems parameterize the parsing model over the set of all possible dependency arcs. The task is to determine, among the exhaustive list of possible arcs connecting any two words in any sentence, weights for the optimal set of arcs that span over all the words and preserve the well-formedness conditions for dependency trees. As such, given that arcs attach dependents to their heads, for graph-based approaches parsing the problem is finding the most likely head for each word, i.e. a head selection problem.

## 5.2 Label First Parsing

Our parsing model is an implementation of the graph based approach. However, it differs critically from the current state of the art implementations of this approach in terms of the order with which it predicts the dependency labels and selects heads.

As we have reviewed in Chapter 3, the state of the art graph based parser implementations usually generate a labeled dependency tree following a *head-first* strategy where labels are predicted for the arcs that have been already identified. Following a *head-first* parsing strategy means the label classifier uses features which represent the properties of the arc $< w_j, w_i >$ as input and tries the figure out the label that best explains the type of the relationship that is predicted to exist between the tokens connected by the arc. Importantly, in this approach the label predictions have no impact whatsoever on how the dependency tree is built. The dependency structure of the sentence is already determined by the head classifier, and dependency labels are employed merely as *names* over an already built set of arcs. In these approaches, the label classifier is usually trained separately than the head-classifier using gold head-dependent pairs from the training data.

There is also a group of studies which follow a *joint* strategy where heads and labels are predicted together. In the *joint* parsing strategy, one would encode the heads and the dependency labels as a set of tuples $< x_i, l_i >$ and the goal of the parser would be to predict these labels of tuples. Theoretically, this design puts equal weight on heads and labels in determining the correct dependency structure of the sentence as it has a joint optimization objective. An important design choice in these works is to figure out how to encode the heads i.e. $x_i$, as part of the labels of tuples $< x_i, l_i >$. Various studies which aim to implement dependency parsing using a sequence-to-sequence model such as Li et al. [63], Spoustova and Spousta (2010) [64], and Kiperwasser and Ballesteros (2018) [65] have dealt with this question. A recent study along this vein, Strzyz et al. [66], have experimented with multiple encoding strategies that consider absolute or relative positions of heads together with their PoS tags, and reported that the one that performs best is using relative positional encodings (distance between the word $w_i$ and the head $x_i$) as well as the PoS tag information of the head $x_i$.

The strategy proposed in this study is *label-first* in the sense we predict the dependency labels over each word before selecting the head for the word. As was explained in the previous chapters, dependency labels are expressions of the grammatical function that a word plays in a sentence. As such, one can easily conceptualize them as properties predicated over words (rather than arcs), representing the words as functions seeking their arguments. In this interpretation, an *nsubj* dependency label represents a word as a function seeking the verbal predicate it takes as an argument, grouping it together with other dependency labels such as *dobj*, *iobj* or *ccomp* in UD label inventory. A *det* relation represents a word as a function seeking its nominal argument, similar to other labels such as *amod*, *nummod*, or *nmod*. As a result, *label-first* parsing is a strategy where we predict the functional category of each word (the dependency label) before determining the function-argument application (the dependency arc) for that word. Conceptually, one can see that this strategy makes dependency parsing closer to Categorial Grammars described in Adjukiewicz (1935) [30] and Bar-Hillel (1953) [31].

Besides its conceptual significance, the choice of whether one employs a *head-first* or *label-first* strategy changes how the information flows within the parsing model, therefore might bear a significance over the parsing performance as well. In this respect, the hypothesis we investigate is whether accurate prediction of grammatical function for each word before predicting its head improves the accuracy of the head selection task for the parser. The motivating idea behind this hypothesis is simple. Knowing the grammatical function of a word early on significantly reduces the search space for its head (or argument), reducing the ambiguity for attachment decisions. For example, an *nsubj* can only attach either to the root of the clause, or to the predicate of one of its clausal complements (i.e. *root*, *ccomp*) depending on whether it is the subject of the matrix clause or not.

What is critically important in the proposed strategy is being able to predict dependency labels without relying on any syntactic clues and simply based on the surface representation of words and their surrounding context. Contrary to *head-first* approaches, in this approach the label classifier does not have access to any arc features $< w_j, w_i >$. While this might not be too much of a problem for dependency labels that have a one-to-one mapping with the lexical categories that they mark (i.e. the *det* label typically marks *a, an, the* in English), the correct grammatical role for many lexical items can only be determined based on surrounding context. For example, knowing whether a nominal functions as a modifier of another noun (*nmod*) or as an argument of a verbal category (*nsubj*) is determined by the surrounding words and the semantics of the verb. Similarly, the correct grammatical roles for words which are verbal complements can only determined by a variety of factors such as case inflection (in languages that have it), positional features (especially in configurational languages) and verb semantics.

Table 5: Baseline metrics for the languages evaluated in the experiments

| Language-Dataset | UAS | LAS |
|---|---|---|
| en-ud | 84.74 | 82.23 |
| en-pud | 88.22 | 85.51 |
| de-ud | 84.10 | 80.71 |
| de-pud | 80.88 | 74.86 |
| fi-ud | 87.97 | 85.64 |
| fi-pud | 90.60 | 88.47 |
| ko-ud | 85.90 | 82.49 |
| ru-ud | 87.15 | 83.65 |
| ru-pud | 82.31 | 75.71 |
| tr-ud | 77.36 | 70.37 |
| tr-pud | 72.33 | 59.57 |
| zh-ud | 74.03 | 68.75 |

### 5.3 Datasets

The ConLL 2017 Shared Task [13] aimed to rank parsers over their performance based on UAS and LAS metrics on multiple languages. As part of the shared task, training and test sets were delivered[67] for the participating teams to evaluate their models, together with word2vec[68] style pretrained word embeddings for each language.

We have run our experiments on 7 languages, namely Chinese (zh), English (en), Finnish (fi), German (de), Korean (ko), Russian (ru) and Turkish (tr). For all the languages except Turkish, we have used the same training and test sets as the ones that were delivered in the CoNLL shared task. For Turkish, we have used a recently developed UD Treebank, the BOUN treebank [69], which is accepted as another state of the art UD treebank for this language. We have also used the same word2vec embeddings that the competing systems used in the aforementioned shared task. Where available, we have evaluated our models on the parallel UD treebank (PUD) corpora that was developed as an additional test set for the same shared task as well.

Stanford's Biaffine Parser [52] which was reviewed in Chapter 3 performed best in the shared task, improving the state of the art metrics across many of the evaluated languages including English, German, Finnish, Korean, Russian, and Turkish. Therefore, in the below experiments, we compare our results to the results reported for this parser. For Chinese, the Stanford Parser was worse than the baseline UD-Pipe model[70] so we take this model as reference. Finally, for Turkish, the Stanford Parser's performance on the newly developed BOUN treebank was reported recently in Özateş (2022) [71], so we take the metrics reported in this study as baseline. Table 5 shows the baseline UAS and LAS metrics for the languages considered.

It is important to note that the selection of the languages which we run experiments on cuts across dimensions such as linguistic typology and dataset size. Among the languages considered, English and Chinese are configurational and analytical languages where the grammatical function of a word is heavily determined by its position in the syntax. These languages typically lack rich morphological inflection or derivation, and do not exhibit the same word order variability that non-configurational languages do. Between them, Chinese is a relatively low resourced language (its training set size consisting of 3497 sentences,) compared to English which has a much larger treebank. On the other hand, Finnish, Russian, Korean and Turkish fall into the class of synthetic/agglutinative languages which are characterized by a relatively free word order and a flat phrase structure. These languages encode grammatical function with case marking, and have productive morphological derivation which increase the vocabulary size considerably compared to analytical ones. Among these languages, Korean and Russian are relatively low resourced (consisting of training set sizes of 4400 and 3850 respectively) compared to Turkish and Finnish (for which the training sets have 7789 and 12217 sentences).

## 5.4 Experiments

We have run two sets of experiments to evaluate the *label first parsing* strategy on the aforementioned datasets. These are described below.

### 5.4.1 Experiment 1: BERT+LSTM Model

#### 5.4.1.1 Description

Since their introduction, pretrained BERT models have proven to be very helpful for many downstream sequence transduction tasks such as Question Answering [72], Text Summarization [73], or Machine Translation [74]. The contextual, non-static nature of BERT embeddings capture semantics of words in different contexts, which has attracted a lot of interest from researchers who wanted to better interpret what kind of linguistic capacities BERT models have. In terms of syntactic capabilities, there have been studies which show BERT's ability to capture subject-verb agreement across a variety of linguistic structures[75]. Various efforts to extract constituency or dependency trees from BERT without any further fine-tuning have concluded that[76][77] structural knowledge is somehow embedded in BERT.

In the experiment that we describe below, we have independently trained two classifiers. First, we have fine-tuned a BERT model on dependency label prediction task without using any arc features to create a *dependency label classifier*. Second, we developed an LSTM based *head-classifier* and pretrained it to predict head-dependent relationships based on inputted dependency labels. At inference time, these two clas-

sifiers are concatenated as schematized in Figure 11[1]. The output predictions from the BERT classifier are passed to LSTM-based head-classifier (together with word embeddings), to predict arcs between heads and dependents.

Note that the architecture of our head-classifier is very much the similar to Stanford Parser except that it does not have the label classification layer on top of the LSTM layers, since predicted labels are already given as input. At inference time, words and dependency label inputs (as predicted by fine-tuned BERT models) are passed through three bidirectional LSTM layers, the outputs states of which are then fed through two ReLU perceptron layers. Similar to the Stanford Parser, these layers create two hidden vectors for each word: one that represents the word as a dependent and the other as head. Finally, the classifier selects the most likely head for each dependent following procedure in equations (11)-(12), which are the same as equations (7)-(8) from Chapter 3.

$$s_i^{(arc)} = H^{(arc-head)} W^{(arc)} h_i^{(arc-dep)} + H^{(arc-dep)} b^{T(arc)} \tag{11}$$

$$y_i'^{(arc)} = argmax_j s_{ij}^{(arc)} \tag{12}$$

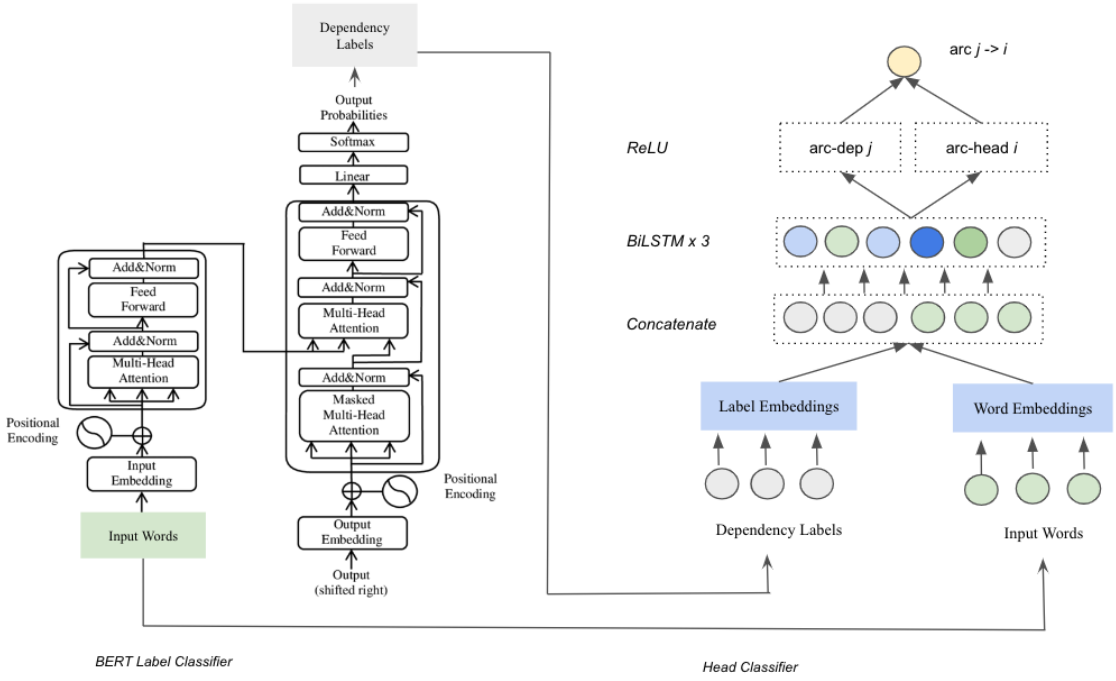


Figure 11: BERT+LSTM model architecture

[1] In Figure 11, the part that represents the BERT Label Classifier is based on the Wikimedia image: *The Transformer Model Architecture*.

Table 6: Hyperparameters for finetuning BERT model

| Param | Value |
|---|---|
| Learning rate | $5e^{-5}$ |
| Optimizer | Adam |
| Dropout | 0.2 |
| Weight Decay | 0.01 |
| $\beta_1$ | 0.9 |
| $\beta_2$ | 0.999 |
| Batch Size | 8 |
| Epoch | 3 |

Table 7: Hyperparameters for training the head-classifier

| Param | Value |
|---|---|
| Word Embedding Size | 100 |
| Label Embedding Size | 50 |
| LSTM Size | 512 |
| LSTM Depth | 3 |
| LSTM Dropout | 0.33 |
| Arc MLP Size | 256 |
| Arc MLP Depth | 1 |
| Optimizer | Adam |
| $\beta_1$ | 0.9 |
| $\beta_2$ | 0.9 |
| Learning Rate | 0.01 |

Another crucial property of the head-classifier is the fact that it is pretrained with an attachment accuracy (i.e. UAS) optimization objective. During this pretraining, besides word inputs, we use gold dependency labels as input features, and we train a label embeddings layer online as part of the training process. This creates a model whose weights are fine-tuned for optimal head classification based on accurate dependency labels. At inference time, we plug in the labels predicted by BERT as input to the head-classifier. The end result is that the more accurate the predicted dependency labels, the higher will be the head-classifier's performance on attachment accuracy.

The BERT models that we fine tune for dependency label prediction are *BERT-base-multiligual-cased* for languages other than English and *BERT-base-uncased* for English. BERT models typically use word-piece tokenization (Song et al. (2020)[78]) for training and inference, which subtokenizes words into smaller units based on a greedy longest-match-first algorithm. On the other hand, baseline parsers that we compare our results with use global embeddings in word2vec style over pre-tokenized datasets. Therefore, to achieve better comparability of results to these parsers, we make sure that the LSTM head-classifier does not rely on BERT's word piece tokens.

Table 8: Label vs. Attachment Accuracy Across Languages and Datasets

| Dataset | Dependency Label Accuracy | Attachment Accuracy |
|---|---|---|
| tr-pud | 79.8 | 74.5 |
| tr-ud | 83.0 | 77.1 |
| ko-ud | 90.1 | 80.6 |
| zh-ud | 90.3 | 83.4 |
| de-pud | 86.6 | 88.2 |
| de-ud | 90.7 | 87.4 |
| fi-pud | 93.4 | 85.8 |
| fi-ud | 92.2 | 85.7 |
| ru-pud | 89.2 | 85.2 |
| ru-ud | 93.4 | 86.5 |
| en-pud | 94.6 | 91.3 |
| en-ud | 94.3 | 88.9 |

In other words, we pass the full form of words (as tokenized in the CoNLL 2017 datasets) separately through word2vec embeddings made available in the shared task, and concatenate them with the dependency label prediction for each word.

Tables 6 and 7 display the parameters employed while fine-tuning BERT for the dependency label classification task and training the LSTM based head-classifier respectively.

#### 5.4.1.2 Results

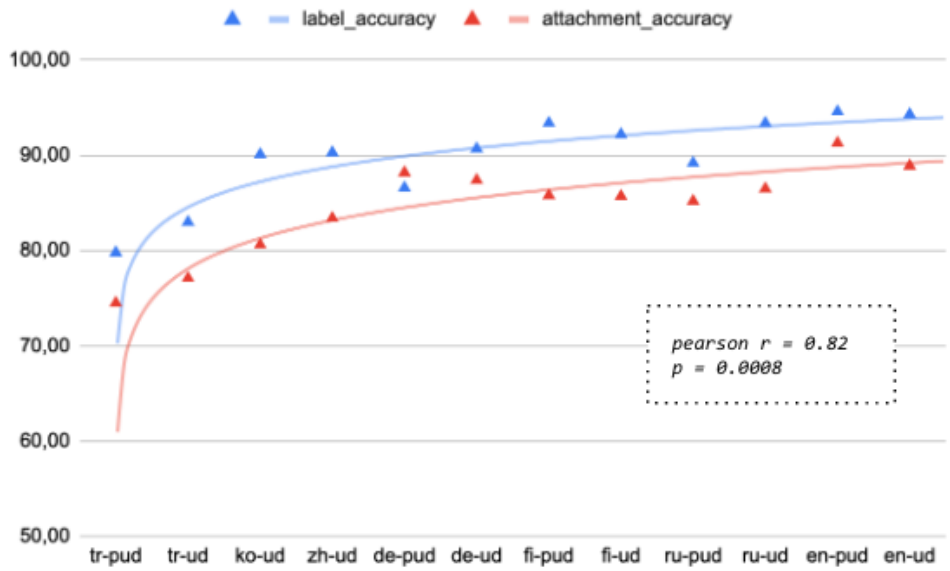


Figure 12: Dependency Label vs. Attachment Accuracy Across Datasets

Table 9: Typological Properties of the Languages in the Experiments

| Languages | en | de | zh | tr | fi | ko | ru |
|---|---|---|---|---|---|---|---|
| Analytic | ✓ | | ✓ | | | | |
| Synthetic/Agglutinative | | ✓ | | ✓ | ✓ | ✓ | ✓ |
| Strict Word Order | ✓ | | ✓ | | | | |
| Case Marking | | ✓ | | ✓ | ✓ | ✓ | ✓ |
| Pro-Drop | | | | ✓ | ✓ | ✓ | ✓ |

In Table 8, we first report the overall dependency label prediction accuracy of the label classifier for each language and dataset, and the attachment scores from the head classifier using the predicted labels as input. The corresponding plot in Figure 12 indicates that there is a strong correlation between label accuracy and head accuracy with a pearson coefficient $r = 0.82$ and $p\text{-}value = 0.0008$, confirming the hypothesis that accurate prediction of grammatical roles early in parsing leads to higher performance in syntactic resolution. When all languages and datasets considered, we see that higher results in dependency label prediction have a positive impact on the dependency parser in terms of finding head-dependent relations. These results set *label first parsing* strategy as a viable strategy for generating labeled dependency trees, one that is promising to provide improvements on state of the art metrics.

It is interesting to inspect the results from the perspective of linguistic typology. In Table 9, we present the linguistic properties of the languages experimented with. On the one hand, we have analytical languages [en, zh] which make very limited use of inflectional or derivational morphology, contrasted with synthetic/agglutinative languages [tr, ru, fi, ko] which heavily rely on morphology for deriving new words, marking arguments (e.g. case-marking), changing valency, and with a rich agreement system. There are languages in between like German which cross-cut a simple binary classification. Even though German does not have as complex a morphological system as languages like Turkish for example, it is still different than English or Chinese in the sense that it has case-declension to indicate grammatical roles and has productive morphological means of word formation using compounding and derivation[2]. Languages with richer morphology and agreement system tend to be pro-drop and display more word order variablity than the ones which do not. In the languages that we experiment with, [tr, fi, ko, ru] falls into this class of languages. Again, German is still a case in between because it is relatively more flexible than English and Chinese in terms of word order, but it does not have pro-drop, and the word order flexibility is not as extensive as languages like [tr, fi, ko, ru][3].

Considering the linguistic typology described above, an interesting pattern emerges in parsing performance where we see higher overall dependency label accuracy in analytical languages with simple morphology [en, zh] than morphologically richer languages [tr, fi, ru, ko], accompanied with higher overall Attachment Score in the

[2] In Parsing literature, German is usually classified as a moderately rich language in terms of morphological complexity (Tsarfaty et al, 2010 [79]

[3] In German, the main restriction on word order flexibility is that the verb cannot change its position.

Table 10: Avg. Label vs. Attachment Accuracy for different language types

| Language Type | Dependency Label Accuracy | Attachment Accuracy |
|---|---|---|
| Analytic (en, zh) | 93.0 | 87.8 |
| Synthetic/Aggl. (tr, fi, ru, ko) | 88.7 | 82.2 |

Table 11: Comparison of current model with the baseline models

| Dataset | UAS (Baseline) | UAS (Current Model) | LAS (Baseline) | LAS (Current Model) |
|---|---|---|---|---|
| en-ud | 84.74 | **88.9** | 82.23 | **86.5** |
| en-pud | 88.22 | **91.3** | 85.51 | **88.2** |
| de-ud | 84.10 | **87.4** | 80.71 | **82.2** |
| de-pud | 80.88 | **88.2** | 74.86 | **79.2** |
| fi-ud | 87.97 | 85.7 | 85.64 | 81.8 |
| fi-pud | 90.60 | 87.9 | 88.47 | 84.2 |
| ko-ud | 85.90 | 80.6 | 82.49 | 76.3 |
| ru-ud | 87.15 | 86.05 | 83.65 | 81.8 |
| ru-pud | 82.31 | **85.4** | 75.71 | **78.6** |
| tr-ud | 77.36 | 77.1 | 70.37 | 68.8 |
| tr-pud | 72.33 | **74.5** | 59.57 | **64.7** |
| zh-ud | 74.03 | **83.4** | 68.75 | **79.7** |

former type of languages compared to the latter. This is shown in Table 10. One possible way to interpret the difference in performance might be that BERT, which is based on an attention based transformer architecture (Vaswani et al., 2017)[80], does not have a notion of word-order except for relying purely on positional embeddings, as was suggested by Goldberg (2019) [75]. Analytic languages, where BERT seems the perform better on dependency label prediction task, are typically the ones where grammatical roles are determined more by syntactic position, as opposed to the other type of languages where inflectional morphology marks grammatical roles and allows for relatively free ordering of lexical items in the sentence. In section 5.5, we will do a more detailed comparative error analysis on one language from each group.

Table 11 compares the performance of label first parsing model against the state of the art performance metrics that were displayed in Table 5. Results where the current model improves the metrics compared to the baselines are boldfaced. As is the practice for evaluating CoNLL datasets, we ignore the *punct* tag while computing results.

The results show that we see a significant improvement compared to the baselines in a number of languages and datasets, with the most significant improvements observed in [en, de, zh] once again, where metrics improve for both UD and PUD consistently. As was noted in the previous section, these are the languages where the label classifier was relatively more accurate in end to end prediction of dependency labels[4]. It is

[4] It is worth mentioning that of course other studies have been made in years following the 2017 shared task on the languages we experiment with here. For example, another shared task was organized in 2018 [81]

also worth observing that an improvement on LAS score is always accompanied by an improvement on overall Attachment Score (UAS), confirming our intuitions that early prediction of accurate grammatical roles, or dependency labels, help improve the overall accuracy of the parser in predicting head-dependent relations.

### 5.4.2 Experiment 2: Joint LSTM Model

#### 5.4.2.1 Description

The previous experiment was limited to testing the *label-first parsing* strategy in a BERT based setting, where dependency labels are independently predicted by BERT disjointly from the head-classifier. An interesting next step could be to compare this model's performance on dependency label prediction to LSTMs which, as opposed to BERT, track states explicitly across the sentence and (arguably) have a better notion of word order than simply relying on positional embeddings [75]. This allows for developing a joint, purely LSTM based architecture which optimize dependency labels and dependency arcs together, following a label-first hierarchy.

In this experiment, we have developed a joint LSTM model which optimizes for dependency labels and dependency arcs jointly in a label-first setting. The model is represented schematically in Figure 13. Specifically, the LSTM model that was used as a *head-classifier* in the previous experiment is expanded with a Softmax Layer that carries out dependency label prediction before the Biaffine classifier performs head-classification. Losses from the two classifiers are summed and backpropagated through the network. As a result, the model creates a pipeline with trainable dependencies between the two classifiers where early stage dependency label predictions determines how the dependency tree is going to be built. The backpropagation of losses from the head classifier to the network, including the Softmax layer, ensures that there is a feedback loop to the label classification layer (in the form of a loss value) from head-classification layer. The idea is that this feedback loop can help tune the model weights to optimize for dependency labels that lead to better attachment decisions.

Contrary to the BERT-LSTM model, we have used the gold PoS tags as input features as well. The PoS Embeddings are trained together with the model, although we do not further fine tune the pretrained word2vec embeddings as part of model training. The LSTM parameters are the same as in Table 7.

---

which included the same set of languages. Therefore, a fair question to ask would be whether these baselines have been improved by different models in more recent years. Unfortunately, the results of the 2018 shared task are not directly comparable to the results we report here because segmentation was also added as another task in 2018 whereas the 2017 shared task was run on already segmented treebanks. Still, let us report for the sake of completeness that the results from 2018 shared task do not show any better results on the languages and datasets we report improvements here. To the author's best knowledge, there are no other studies that improve the dependency parsing accuracy for the relevant languages more than the currently reported improvements using the exact same features/embeddings on the same datasets.

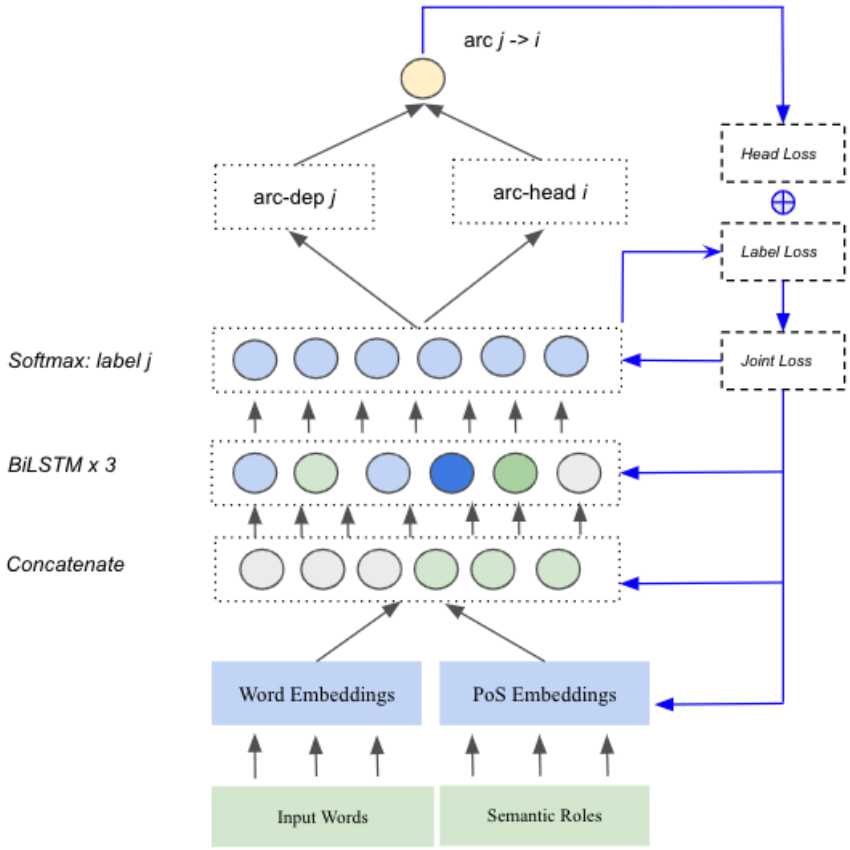


Figure 13: Joint LSTM model architecture

#### 5.4.2.2 Results

We have run this experiment on 3 languages where BERT+LSTM model relatively underperformed in Experiment 1 (Russian, Korean and Turkish) and one language where BERT+LSTM overperformed the state of the art in Experiment 1: Chinese. The goal here is to see whether a fully LSTM based architecture, with a joint optimization objective performs as good as or even better than a disjoint BERT based architecture in different language types.

In Table 12, we compare the results from the joint-LSTM model to results of the model described in the previous experiment, reporting Label Accuracy, Attachment Accuracy (UAS) and Labeled Attachment Accuracy Scores (LAS). Note that these results are based only on UD sets (i.e. non-PUD sets).

In all the languages, we observe the BERT+LSTM model outperforms the joint model across all metrics. The Label Accuracy metrics indicate that the contextual embedding that BERT employs is much more powerful than the LSTM model in figuring out correct grammatical roles in the sentence. The results also further confirm the hypothesis that one can constrain the dependencies with accurate prediction of gram-

Table 12: Comparison of BERT+LSTM model with joint-LSTM model

| Model | Language | Label Accuracy | Attachment Accuracy | LAS |
|---|---|---|---|---|
| **Bert+LSTM** | ru | 93.4 | 86.5 | 81.8 |
| Joint-LSTM | ru | 83.9 | 78.0 | 69.5 |
| **Bert+LSTM** | ko | 90.1 | 83.4 | 76.3 |
| Joint-LSTM | ko | 71.9 | 68.9 | 60.0 |
| **Bert+LSTM** | zh | 90.3 | 83.4 | 79.7 |
| Joint-LSTM | zh | 80.3 | 73.2 | 64.9 |
| **Bert+LSTM** | tr | 83.0 | 77.1 | 68.8 |
| Joint-LSTM | tr | 79.1 | 74.3 | 64.9 |

matical roles at an earlier stage during parsing. For all the languages in Table 12, the drop in Attachment Accuracy (compared to results in Experiment 1) is the result of the drop in Label Accuracy.

### 5.5 Error Analysis

In this section, we perform error analysis on English and Turkish based on the BERT-LSTM model performance over the PUD test set. We focus more specifically on errors related to predicate-argument structure, and try to explain the difference in performance between the two languages. We have shown in Table 11 that for both languages, the label first BERT-LSTM model over-performed the state of the art on the PUD sets. However, when the two languages are compared, there is a big difference in performance (*tr=UAS: 74.5, LAS: 64.7* vs. *en=UAS: 88.9, LAS: 86.5*). As we have also shown, the overall label accuracy of the English model was much higher than the Turkish model (94.3 vs. 79.8) correlating with the higher performance on syntactic attachment.

The use of PUD gives us an opportunity to compare the two models over sentences that have the same semantics as this corpus contains exact translations of the same sentences across languages. As was noted before, these treebanks were developed as an additional test set for the ConLL 2017 shared task where sentences sourced from English Wikipedia were translated to other languages and annotated by linguists of the relevant language.

Following Can et al (2022)[82], we first look at how robust the two models are to longer sentences. Afterwards, we deep dive into the type of errors that are relevant to predicate-argument structure for the Turkish set.

### 5.5.1 Effect of Sentence Length

The performance of the two languages with respect to varying sentence length is plotted in Figures 14 and 15. Sentence length is determined by the number of tokens in the sentence. We kept a sentence length interval as 10 and plotted the UAS and LAS metrics in sentences of length up to 50.

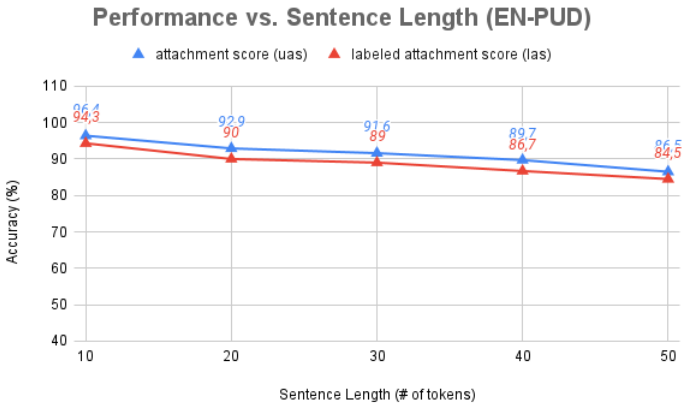


Figure 14: English BERT-LSTM Model's Performance by Sentence Length

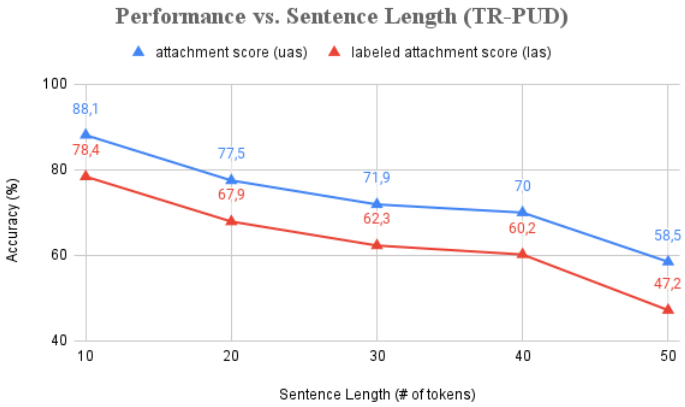


Figure 15: Turkish BERT-LSTM Model's Performance by Sentence Length

Lengthier sentences imply more complicated predicate argument structures as they are more likely to include multiple clauses in the sentence and longer distance dependencies. The results show that the Turkish model is much less robust to increasing sentence length compared to the English model. While in the English model, Attachment Accuracy between shortest sentences ([1-10) tokens) and longest sentences ([41-50) tokens) only drops by ~10%, for Turkish it drops for ~30%. Similarly for Labeled Attachment Scores.

As we are following a *label-first* strategy, it would be interesting to see how these scores correlate with the respective label accuracies of the two models. For English,

we would expect that the label classifier did a better job in identifying the grammatical roles in relatively larger sentences compared Turkish. This is plotted in Figures 16 and 17.

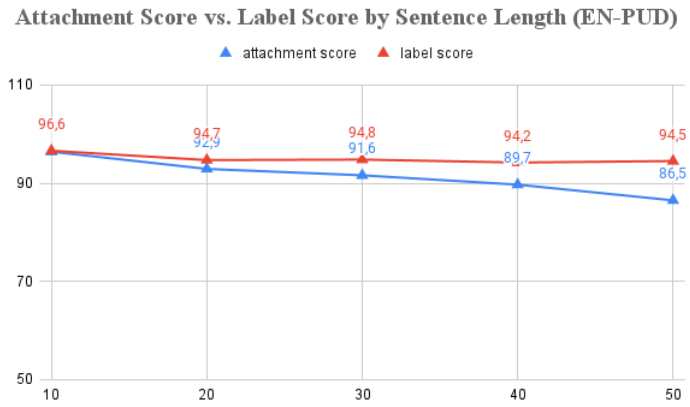


Figure 16: Attachment vs. Label Accuracy, en-pud

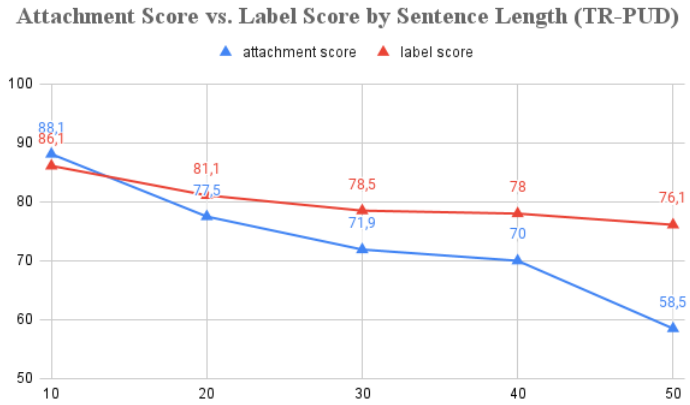


Figure 17: Attachment vs. Label Accuracy, tr-pud

As can be seen from the plot in Figure 16, the English model much more consistent in accurately predicting grammatical roles even in longer sentences. By contrast, in the Turkish model, the continuous drop in predicting grammatical roles in longer sentences seems to correlate with the drop in predicting syntactic dependencies accurately.

The results so far confirm the hypothesis about *label-first* parsing which argued that accurate prediction of grammatical roles lead to accurate discovery of syntactic structure. Based on comparative evaluation, we have seen that languages that perform consistently high in predicting dependency labels first perform better in resolving the syntactic relations in the sentence, even in longer sentences.

Table 13: Arguments and Adjuncts in UD

| | Subject | Object | Adjunct |
|---|---|---|---|
| **nominal** | nsubj | obj/iobj | obl |
| **clausal** | csubj | ccomp | advcl |
| **open** | | xcomp | |

In the next section, we will do a deep dive into the type of errors that are directly related to verbal predicate argument structure. We are going to evaluate how the models perform in determining correct predicate-argument structures in different contexts, especially in terms of argument-adjunct distinction and in sentences with embedded clauses.

### 5.5.2 Errors in Verbal Predicate Argument Structure

As we mentioned in Chapter 4, UD defines 8 tags in total to mark grammatical arguments and adjuncts in the sentence. These are repeated in Table 13 above for convenience. We consider an error in parsing a *predicate-argument structure* error if it is an error that concerns one of the grammatical role labels in Table 13. These predicate-argument structure errors can be further subdivided into three categories:

1. **Grammatical Role Errors**: The parser correctly identifies the predicate that the word should attach to, but misclassifies its grammatical role, i.e. a *subject* argument of a verb is misclassified as *object*.
2. **Attachment Errors**: The parser correctly identifies the grammatical role of a word, but wrongly identifies its predicate, i.e. a *subject* is attached to a wrong predicate.
3. **Role-and-Attachment Errors**: The parser misclassifies both the grammatical function of a word and its head.

First, in Tables 14-15, we look at how many argument structure errors there are for each language and how they are distributed per grammatical role in the two languages. Argument structure errors account for a much larger portion of total errors Turkish (~34%) compared to English (~19%), implying that resolving the verbal complements and adjuncts is a harder task for the Turkish model. In terms of the distribution of these errors over grammatical roles (Table 15), a few observations seem to be emerging.

First, both languages seem to be doing relatively well on identifying clausal dependents (*xcomp, ccomp, csubj*). In fact, the total portion of errors concerning clausal dependents is even lower in the Turkish model than its English counterpart. The reason behind this can be that in Turkish the (non-finite) subordinate clauses are explicitly

Table 14: Errors in Predicate-Argument Structure in En and Tr (PUD sets)

| language | total tokens | total errors | total argument structure errors |
|---|---|---|---|
| **tr** | 16881 | 6136 | 2075 (%33.7) |
| **en** | 21176 | 2726 | 508 (%18.8) |

Table 15: Distribution of Predicate Argument Errors Per Grammatical Role

| Label | En (%) | Tr (%) |
|---|---|---|
| **nsubj** | 18,70 | 22,41 |
| **obl** | 34,65 | 28,53 |
| **advcl** | 16,54 | 11,76 |
| **ccomp** | 8,46 | 3,04 |
| **xcomp** | 9,25 | 5,73 |
| **csubj** | 2,56 | 2,41 |
| **obj** | 8,86 | 19,47 |
| **iobj** | 0,98 | 6,65 |

marked by morphology *-mAK, -mA, -DIK, -(y)AcAK or -(y)Iş)* . Even though we did not apply any further morphological preprocessing or used morphological labels during determining grammatical roles, it is very likely that the word piece tokenization BERT models employ have developed sensitivity to these structures.

Second, both languages seem to be struggling with adjunct roles (*advcl, obl*), and specifically obliques. In the Turkish data, the *obl* tag was confused with a variety of tags, but mostly with adverbial clause modifiers (*advcl)*. In both Turkish and English, it was also confused with nominal modifiers (*nmod:poss* and *nmod*) with very high frequency.

Next, we look at the distribution of error types that were defined above for each of the argument and adjunct categories in Turkish data and inspect some interesting examples. Table 16 illustrates the distribution of Attachment and Grammatical Role Errors across categories. We have also reported a sub-type of attachment errors, i.e. attachment errors in a multi-clause sentences. These are errors where the system attached the relevant argument to a wrong predicate even though it predicted the grammat-

Table 16: Distribution of Error Type Per Grammatical Role

| Error Type | obl | nsubj | obj | advcl | iobj | xcomp | ccomp | csubj |
|---|---|---|---|---|---|---|---|---|
| Attachment (%) | 53,58 | 40,98 | 26,56 | 41,79 | 0 | 0,84 | 16,42 | 13,73 |
| In Multiclause Const. (%) | 9,11 | 10,02 | 4,43 | 14,93 | 0 | 0 | 4,48 | 5,88 |
| Role (%) | 11,79 | 14,21 | 29,18 | 21,64 | 57,97 | 58,82 | 44,78 | 41,18 |
| Both (%) | 34,63 | 44,81 | 44,27 | 36,57 | 42,03 | 40,34 | 38,81 | 45,1 |

Table 17: Confusion Table for Clausal Dependents

| **category** | **confused tag** | % |
|---|---|---|
| ccomp | obj | 56,67 |
| | obl | 16,67 |
| | amod | 13,33 |
| | nsubj | 3,33 |
| | acl | 3,33 |
| | advcl | 3,33 |
| xcomp | ccomp | 34,29 |
| | advcl | 21,43 |
| | obl | 17,14 |
| | obj | 12,86 |
| | nmod:poss | 1,43 |
| | csubj | 2,86 |
| | amod | 4,29 |
| | nummod | 2,86 |
| csubj | nsubj | 66,67 |
| | obj | 23,81 |
| | amod | 4,76 |
| | acl | 4,76 |

ical role of the argument correctly. We analyze some of the errors in detail in the next section and try to understand what they mean for the current implementation and learnability of predicate argument structure.

***Grammatical Role Errors***

When the *role errors* are considered, interesting to inspect are the clausal dependents *xcomp*, *ccomp*, and *csubj*, which seem to account for the highest number of errors in this category. Notably, this is reversely correlated with the lower amount of Attachment Errors for these labels. This signals that even though the system mispredicted the grammatical role for the lexical items that should bear this role, it still managed to find the right head in many cases for those items.

Given the *label-first* parsing approach that is employed in this study, this would happen mostly if the confused category for these labels is also one that is the same type of verbal complement. In Table 17, we present the confusion table for these categories, which seem to confirm our hypothesis. The *ccomp* category is mostly confused with *obj*, meaning that the system managed to understand the object of the sentence, but did not realize the object was a clause. Similarly, the *xcomp* category, which stands for non-finite clausal complements with open argument position, is mostly confused with another clausal complement category *ccomp*, and the *csubj* is confused mostly with another Subject category *nsubj*. In all these cases, it seems that the system managed to identify the argument roles correctly, leading to accuracy in figuring out syntactic dependencies. We believe these results can be directly attributed to the properties of the

Table 18: Confusion Table for Adverbial Clauses

| **category** | **confused tag** | % |
|---|---|---|
| advcl | obl | 36,21 |
| | ccomp | 22,41 |
| | amod | 6,9 |
| | advmod | 6,9 |
| | obj | 4,45 |
| | nsubj | 1,72 |
| | nmod | 2,0 |
| | aux | 1,72 |

current parsing strategy. As we have described section 5.4.1, Figure 11, during training we embed the dependency labels into vector space. The embedding mechanism leads the system to learn how to cluster similar dependency labels together based on their distributional and syntactic properties. We take this as a factor explaining why the current implementation leads to higher attachment accuracy across a range of languages. Even though the system might confuse the exact grammatical role, this does not always lead to wrong attachment if in most cases it is confused with a role that has similar syntactic properties.

The more severe cases in Table 17 are the second mostly confused grammatical categories: *xcomp->advcl*, *ccomp->obl*, *csubj->obj*. All these cases mean that the system has confused the predicate-argument structure of the verb in question, classifying clausal complements either as adjuncts when they are objects or objects when they are subjects. Similarly, when we look at the confusion table of *adverbial clauses (advcl)*, we observe an important ratio of mistakes are due to confusing them with the argumentative role *ccomp*, which can also be classified as an argument-adjunct distinction error.

Without having a baseline which also reports predicate-argument structure errors in the way we do, it is difficult to state how well the current implementation fares against different implementations in resolving predicate-argument structure in dependency parsing. However, looking at the confusion tables for argument and adjunct roles, one positive side of the current strategy is that most of the confusion happens between grammatical categories that have the same argument roles (i.e. *ccomp->obj*, *xcomp->ccomp*, *csubj->nsubj*, *advcl->obl*). This can be interpreted as a positive impact of the current implementation which tries to fine-tune dependency labels for accurate dependency parsing.

***Attachment Errors***

Interesting in the Attachment Errors are those that include multi-clause sentences or sentences with embedded clauses. These are errors where an argument, even though correctly identified to be a subject or object, might be attached to the wrong predicate (e.g. a subject is confused by the system to be the subject of the embedded clause

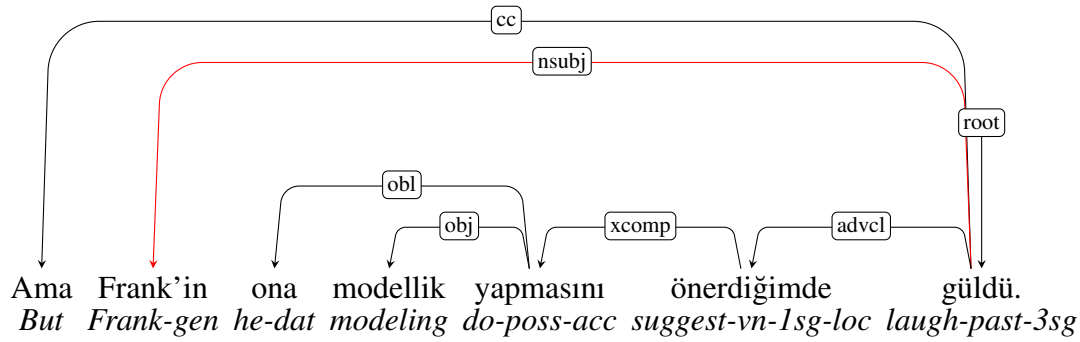


Figure 18: Dependency tree of sentence *'Ama Frank'in ona modellik yapmasını önerdiğimde güldü.'*

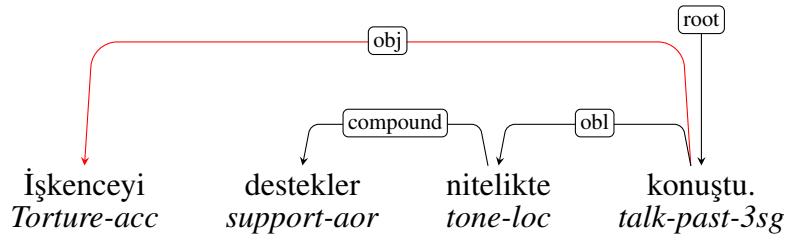


Figure 19: Dependency tree of sentence *'İşkenceyi destekler nitelikte konuştu.'*

rather than the main clause). Two examples from Turkish eval data where the model has made such errors are given in Figures 18 and 19.

Errors in both 18 and 19 can be attributed to the lack of parser's ability to figure out the argument structure for the verbs in question. In Figure 18, there are three embedded clauses and only the innermost one has an overt subject. The model attaches this subject to the root of the main clause *laugh* rather than the embedded one. In Figure 19, the verb *destekle-* (support) requires an internal object while the verb *konuş-* (talk) does not. However, the model confuses the predicate-argument structure of the two verbs and chooses the the main verb *konuş-* to be the head of the object *işkenceyi*.

These examples are illuminating in terms of why it is more difficult to learn predicate argument structure from dependency treebanks for certain languages. As we see in both examples above, in languages like Turkish, certain arguments might be covert in the surface representation due to agreement morphology over the predicate which helps them be understood at the semantic level. In such cases, the model can only account for such unseen data if one of the two conditions hold: a) if it has seen the relevant predicate appearing in all possible argument combinations in the dataset already and can therefore predict that the language allows for certain arguments to be missing in the surface structure b) if the parser has prior knowledge about the subcategorization requirements of the verb, and can therefore understand from the sentence which argument is missing at the syntactic level.

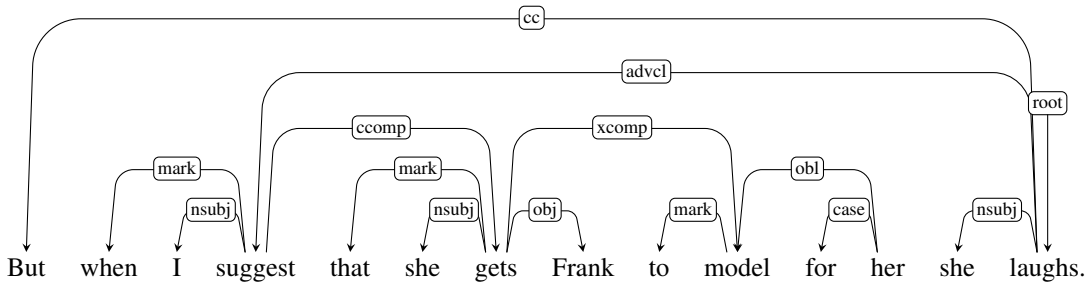


Figure 20: Dependency tree of sentence *'But, when I suggest that she get Frank to model for her, she laughs.'*

It is unrealistic to expect that condition (a) can be true for most of the predicates in any treebank. On the other hand, one can presume that due to the linguistic properties of the langauge, a Turkish treebank (or treebanks of similar languages) will always be more sparse than its English counterparts in terms of its richness for exposing different predicate-argument combinations to the model. What we see in Figures 18 and 19 is that this sparseness easily leads to model bias. In both examples, the model is overly greedy for attaching subjects or objects of the embedded clauses to the *root* of the main clause. Given the data sparseness problem, the only way to overcome these type of errors for accurate parsing, especially for languages like Turkish, is to decorate lexical-functor categories richly with lexical semantic information such as valency or subcategorization. This would allow for the model to have prior knowledge that a predicate like *konuş-* (talk) does not call for a direct object argument while a verb like *destekle-* (support) does, therefore helping the model score the parse in Figure 19 as less likely in the context of these verbs.

Compare the parse in Figure 19 to the English model's parse of the same sentence in Figure 20 where we can see that the English model predicts the predicate argument structure of all the verbs correctly. An important point to note here is that the analytical nature of English requires all arguments of a predicate to be overtly represented in the surface structure. This makes the datasets for languages like English less sparse, therefore providing the opportunity for the parser to better learn subcategorization directly from treebanks even when such information is not encoded on the lexical items. We argue that this is one of the main factors that explains the variability in performance between analytical languages like English and synthetic/agglutinative languages like Turkish in terms of learning predicate-argument structure from dependency data. In analytical languages like English which have poor morphology, arguments are never covert, so a parser is more easily exposed to relevant predicate-argument combinations for a verb in the surface representation. If we want comparable performance in languages like Turkish we need to deal with the data sparseness that is inevitably caused by the linguistic properties of the language. One way to deal with this is to annotate the lexical-functor categories with information relevant to predicate-argument structure[5]. This way, we can train a dependency parser

[5] For example, similar to how CCGs which were reviewed in Chapter 2 represent the predicate-argument structure of functor categories.

in such a way that derivations that does not satisfy the selectional requirements of the predicates are ranked lower than those that do.

## 5.6 Conclusion

This chapter constitutes the heart of this thesis. In this chapter, we have developed a novel parsing strategy called *label first parsing*. We have shown that a BERT based implementation of this model improves the parsing accuracy across a number of languages and baselines. This was followed by an error analysis, particularly on Turkish and English examples. There are a number of conclusions that can be drawn from this chapter.

First, we see that the *label-first* parsing strategy improves state of the art metrics for quite a few languages due to the fact that it prioritizes identifying predicate argument structure first in a dependency parsing problem. Accurate identification predicate and argument roles makes parsing a much simpler task for the model as it considerably reduces the ambiguity concerning how the syntactic dependencies should be established. As part of this parsing strategy, we have shown that dependency labels can be interpreted as properties predicated over words, defining them as functions seeking their arguments (or heads). We have mentioned that such an approach makes dependency parsing closer to Categorical Grammars. Finally, we have shown that the developed parsing strategy allows for clustering syntactically similar dependency labels together during training, which positively impacts the overall attachment accuracy for a parser.

Second, the comparative error analysis on Turkish and English shows that the dependency formalism is better suited for inducing predicate-argument structure in certain type of langauges than others. The dependency framework relies purely on surface representation and have no means to account for data that is relevant to predicate-argument structure at the semantic level but not visible in the surface. This property of dependency grammars hinders learnability of predicate-argument structure from dependency treebanks more drastically in synthetic/agglutinative languages where data can be more sparse due to morphological complexity which might cause arguments be covert in the surface structure. We have shown that this is an important factor that explains variablity in parsing performance between English and Turkish on the same dataset. The main argument that we derived from the results we obtained is that to be able to learn predicate-argument structure for languages with different typological characteristics from data, one needs to encode lexical semantic properties of predicates such as their subcategorization and selectional restrictions on the lexical items. Otherwise, datasets of certain langauges will inevitably be more sparse due to typological reasons, which will lead to poorer performance in learning and parsing.

# CHAPTER 6

# LABEL FIRST PARSING WITH SEMANTIC ROLES

## 6.1 Introduction

In the previous chapter, we have shown that using a dependency parsing methodology that starts from predicting grammatical roles and builds a syntactic tree based on them improves parsing performance for a number of languages. Importantly, even in languages where we have not seen improvements relative to the baseline parsing models, we have seen a strongly positive correlation between a parser's performance in predicting grammatical roles and resolving for syntactic dependencies. We have discussed that this is because grammatical roles constrain the hypothesis space for dependencies, reducing considerably the attachment ambiguity for a parser.

It has also been discussed in that Chapter 2 that from a linguistic perspective which is assumed in a large body of theories dealing with syntax-semantics interface, grammatical roles are semantically derived rather than being primitive constructs of grammar. Works like Jackendoff (1974) [35], Talmy (1985) [15], and Levin and Rapaport Hovav (2005) [23], among many others, have all argued for a multistratal theory of grammar where semantic roles that exist in the Lexical-Conceptual Structure of a verb are mapped to grammatical roles in the sentence based on various argument realization principles. Following these ideas, in this chapter we design an experiment where we start parsing not directly from grammatical roles but from semantic roles, and aim to predict grammatical roles based on them. The hypothesis is that a parser that has access to semantic role information can learn how to map semantic roles to grammatical roles, which should improve the accuracy of grammatical role labeling, which, in turn, should lead to more accurate resolution of syntactic dependencies. Therefore, the goal is to show if, or how much, such a hierarchical set up that uses semantic role labels leads to better learning of syntactic dependencies for a parser.

Note that, unlike the experiment in the previous chapter where we learned both grammatical roles and dependencies, in this experiment we do **not** also aim to learn semantic roles during training, but treat them as given. In practice, this means we are using the gold semantic role annotations as features while learning grammatical roles and dependencies, rather than trying to optimize the parser for predicting them as well. This is because the main contribution of this experiment is not about finding out ways to engineer a model that can best optimize for the relevant tasks involved. Rather, it has a more linguistically driven concern of investigating in what ways, if

at all, knowledge of semantic information in the form of thematic role labels leads to reduction in predicate-argument structure errors for a dependency parser. Therefore, the main focus will be given to analysing the errors that the parser makes or recovers from in this set up compared to one which does not use semantic role labels. If it can be shown that such a parser really makes better generalizations about predicate argument structure, future work or experiments can be devised to investigate how to efficiently optimize for all the tasks also involving semantic role labeling, in a hierarchical, multitask setting. Also note that contrary to previous chapter, current experiment is only run on Turkish data.

In section 6.2., we first start with reviewing the dataset we will use in this experiment, i.e. the Propbanks, which are dependency treebanks extended with semantic role labels.

## 6.2 Propbank and Semantic Dependencies

Since their introduction to grammatical theory, thematic roles (or semantic roles) have had a prominent role in understanding the syntax-semantics interface and have been employed to theorize how semantic information is mapped into grammatical structure. Every verb expresses an *eventuality* and eventualities typically contain participants; individuals involved in the event or the state expressed by the eventuality. A thematic role is a conceptual representation of how exactly a participant is involved in an eventuality: whether a participant is doer of the action (AGENT), the undergoer of the action (PATIENT), the perceiver of the action (EXPERIENCER) and so on. As was shown in Chapter 2, many grammatical theories assume these roles exist as arguments in the lexical-conceptual structure (LCS) of verbs, and represent them usually with $\theta$-grids.

Thematic roles are a distinct layer of representation than grammatical roles. While the latter is a syntactic notion concerning how an argument fits into the *subcategorization frame* of a verb to make a syntactically well-formed sentence, the former is a semantic notion concerned with what kind of semantic properties this argument should possess. On the other hand, as many studies have successfully shown, thematic representation is one which syntax is sensitive to and one which can constrain syntactic behaviour. For example, we know that if a verb defines a participant role that causes an action and another participant role that undergoes a change in its $\theta$-grid, the former is realized as the subject while the latter as the object. This and similar kind of mapping principles explain the behaviour and lexicalization patterns of many verbs, positing generalizations about how a verb *realizes* its arguments in syntax based on its lexical-conceptual structure.

Propbanks (Kingsbury and Palmer, 2002) [83], (Palmer et al., 2005)[84] are extensions of dependency treebanks where semantic roles associated with each verb in the sentence is explicitly marked over the grammatical arguments. This enriched representation provide a way to illustrate, for each argument in the sentence, the thematic role that it fulfills with respect to a verb as a semantic dependency, together with how

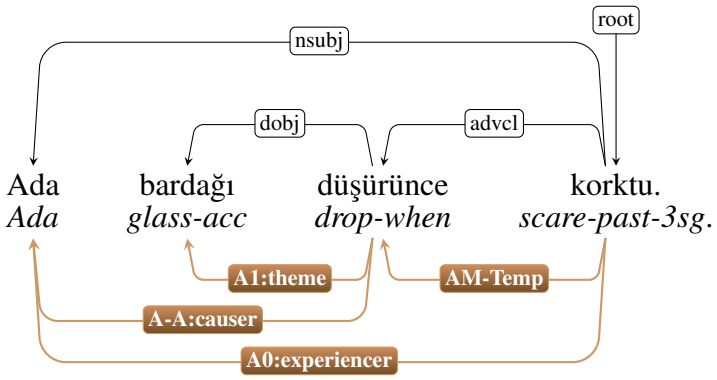


Figure 21: Semantic tree of sentence *'Ada bardağı düşürünce korktu.'*

this thematic role is realized into a grammatical role in the syntactic dependency. The ability to see both syntactic and semantic roles makes it possible to better represent linguistic relations in sentences where a single argument might fulfill multiple roles relevant to different predicates, for example in a sentence like Figure 21.

Here, *Ada* is both the experiencer of *kork-* (be afraid) and the causer agent of *düşür-* (drop), since both of these verbs define these theta roles in their LCS which should be preserved in the syntactic representation. Of course, theories like GB (Chomsky, 1981)[17], guided by the theta-role assignment principle, would not represent these distinct thematic roles over the same argument but use a dummy PRO argument which would be assigned the AGENT role of the verb in the embedded clause. However, as was the case in UD treebanks, UD Propbanks try to remain theory agnostic. The adjunct status of the adverbial clause is represented in the semantic tree with the role *AM-Temp*, marking manner and temporal adjuncts.

Looking at the tree in Figure 21, one can see that the labels Propbanks use are not exactly the same as the familiar thematic role lists. In fact, here, we have added the thematic role labels such as *agent*, *theme* ourselves while the actual labels used in Propbank are *ArgA, Arg0, Arg1* and so on. However, the Propbank guidelines state that they are designed with thematic roles in mind. Below, in Table 19, we give definitions of each of the Propbank labels, together with their counts in the Turkish Propbank. The counts are taken from (Şahin, 2018: 55) [85] and are the sum of all the labels in *training*, *dev* and *test* sets. The argument definitions are based on the current Propbank annotation guidelines (Bonial et. al, 2012) [86].

Looking at the Semantic Role Labels, one can see that actually Propbanks define 7 different semantic roles, namely; A0-4, A-A, and variations of AM. The roles A0-4 correlate directly with thematic roles, while the A-A tag is used to represent causer-agents in morphologically derived causative constructions (e.g. *(-DIR)*) in Turkish.

Table 19: Semantic Role Labels in Turkish Propbank

| Label | Definition | Count |
|---|---|---|
| **A0** | Agent, Experiencer | 3800 |
| **A1** | Patient, Theme | 7812 |
| **A2** | Instrument, Benefective | 1330 |
| **A3** | Source, Benefactive | 289 |
| **A4** | Ending point | 621 |
| **A-A** | Secondary agents or Causer Agents | 178 |
| **AM-TMP** | Temporal modifier | 1614 |
| **AM-MNR** | Manner adverbs | 1486 |
| **AM-LOC** | Locative modifier | 859 |
| **AM-LVB** | Light verb | 694 |
| **AM-GOL** | Goal | 416 |
| **AM-EXT** | Extent modifier | 384 |
| **AM-CAU** | Cause Clauses | 359 |
| **AM-ADV** | Adverbials (which don't fit to any other labels) | 317 |
| **AM-INS** | Instrument | 167 |
| **AM-PRD** | Secondary predication | 144 |
| **AM-DIS** | Discourse | 139 |
| **AM-TWO** | Reduplicative Adjuncts | 92 |
| **AM-COM** | Comitative modifier | 84 |
| **AM-DIR** | Directional modifiers | 78 |
| **AM-NEG** | Negation | 67 |
| **AM-MOD** | Modal Verb | 6 |
| **AM-REC** | Reciprocals | 1 |

## 6.3 Experiment

In this section, we first do a brief review of works that try to integrate semantic role labeling into dependency parsing. Such works can be categorized into two main approaches. In the first group, represented by studies such as Punyakanok et al (2008) [87], Gildea and Palmer (2002)[88], and Gildea and Jurafksy (2002)[89], the main objective is to learn semantic roles using features from dependency (or constituency) trees. These works employ a pipeline model where input sentences are first parsed into syntax trees, from which various linguistic features are extracted to help an SRL model. A technical variation of this approach can also be found in Che et al (2008)[90] and Johansonn and Nugues (2008)[91], where the main idea is to use a k-best list of candidate dependency trees which is reranked based on a global SRL model. The common idea among all these studies is that syntactic priors are crucial for accurate semantic role labeling.

The second group of works follow a joint strategy by which they to optimize a model for syntactic and semantic dependencies together. A challenging aspect of such an approach is the fact that in the dependency representation the syntactic and semantic layers are not isomorphic. In Propbank data, for example, dependencies in the syntactic layer are bilexical, meaning one argument can only be linked to one head. On the other hand, in the semantic layer more than one semantic role label can be defined over the same argument. Even though this is not a problem from a purely representational perspective, this divergence between the two layers make the commonly used arc-factored models [47] unsuitable for the semantic parsing task.

A common approach devised to overcome the divergent representation is to preprocess the data to make it suitable for joint processing. This approach has been employed by Musillo and Merlo (2006)[92], Yi and Palmer (2005)[93], and more recently by Shi et al (2020) [94] . The trick is to create joint-tags by enriching dependency labels with semantic role labels. For example, an *nsubj* tag would have a list of variants such as: *nsubj-A0, nsubj-A1* and so on which represent grammatical and semantic role labels together. Of course, from a computational perspective, this approach brings in more data sparsity problems as it leads to a proliferated number of tags where most of them will have few or no training examples.

Even though we do not jointly learn the SRL task as part of our experiment, it would be fair to say that the proposed architecture is different from the aforementioned set of works in the sense that we start from semantic role labels to build up a syntax tree. This is an extension of the label-first parsing strategy that was proposed in the previous chapter. The goal is to see to what extent access to semantic information helps a label-first parser to better identify grammatical roles and dependencies and how it translates into a reduction in predicate-argument structure errors. As such, this experiment and analysis can be considered as a precursor to further experiments which might use the same hierarchical parsing strategy while at the same time jointly learning both semantic and grammatical roles as well as dependencies.

### 6.3.1 Limitations of the Data

Before talking about the experiments, we should briefly talk about the limitations of the data that we use in this chapter.

The experiments in the previous chapter were based on treebanks that use the latest version of UD (v2.9) for all the languages. However, the Turkish Propbank data [85] which we run the experiments of this chapter on, is developed based on an earlier version of UD which has an impoverished number of grammatical role labels. Specifically, the *advcl* tag does not exist, and all kinds of adjuncts, whether nominal or adverbial, are annotated with the *obl* tag. Furthermore, the *xcomp* tag does not exist, and all kinds of clausal complements are mapped to the *ccomp* tag. Finally, Propbank does not distinguish between indirect objects and direct objects and use the *obj* tag to cover both.

There are also distributional problems in the Propbank data. For example, the *ccomp* tag only appears 13 times in the test and the dev sets combined, meaning that the treebank's eval set only has 13 sentences with clausal complements. The situation in clausal subjects is similar, which appear only 2 times in the Propbank's test set, and only once in the dev set. The dependency labels in the Propbank test and dev sets together with their counts is presented in Table 20.

Since the goal of the experiment here is to analyse how semantic role labels help identify correct grammatical roles for a dependency parser using Propbank data, we should acknowledge that the impoverished amount of grammatical role labels as well as the distributional problems associated with this data impedes a very detailed analysis. Therefore, when we discuss how much Propbanks help with learning predicate-argument structure in what follows, we will be able to focus mainly on nominal subjects and objects and the oblique adjuncts as much as possible.

### 6.3.2 Model

For the experiments, we have used the exact same model that was described in chapter 5, Figure 13, with the same hyperparameters that were defined in Table 7 in the same chapter. This is the joint LSTM model that tries to optimize dependency labels and dependency heads together. On the base side, we have trained this model only with word embeddings and PoS tags while on the experiment side we have also used semantic role labels. As we have mentioned, the goal of the experimental model is to see whether semantic role labels help improve grammatical role accuracy and, in turn, syntactic dependencies. From now on, we will refer to the model that did not use any semantic role labels as the *baseline* model and the other one as the *experimental* model.

Due to the small size of Propbank data, we have used both the test set and the dev set for our training and evaluations. That is to say, in one case we have merged the

Table 20: Dependency Labels in Propbank with Counts

| **category** | **Test Set Count** | **Dev Set Count** |
|---|---|---|
| acl | 276 | 256 |
| advmod | 282 | 287 |
| advmod:emph | 147 | 144 |
| amod | 532 | 560 |
| appos | 6 | 8 |
| aux:q | 47 | 33 |
| case | 351 | 374 |
| cc | 126 | 153 |
| ccomp | 7 | 6 |
| compound | 303 | 311 |
| compound:lvc | 92 | 79 |
| compound:redup | 37 | 32 |
| conj | 580 | 614 |
| cop | 131 | 114 |
| det | 308 | 296 |
| discourse | 25 | 34 |
| fixed | 15 | 15 |
| flat | 175 | 168 |
| mark | 14 | 16 |
| nmod | 532 | 533 |
| nmod:poss | 615 | 560 |
| nsubj | 600 | 582 |
| nummod | 99 | 70 |
| obj | 669 | 707 |
| obl | 765 | 762 |
| parataxis | 1 | 4 |
| punct | 1592 | 1607 |
| root | 869 | 865 |
| csubj | 2 | 1 |

Table 21: Overall Parsing Accuracy of the Baseline vs. Experimental Model

| | **Baseline** | **Experimental** |
|---|---|---|
| UAS (Dev Set) | 73.8% | 76.6% |
| LAS (Dev Set) | 58.5% | 62,5% |
| UAS (Test Set) | 73.6% | 76,8% |
| LAS (Test Set) | 59.9% | 63,7% |

Table 22: Performance in Identifying Grammatical Roles in Base vs. Exp. Model

| | **Precision (Dev)** | **Recall (Dev)** | **Precision (Test)** | **Recall (Test)** |
|---|---|---|---|---|
| nsubj (Baseline) | 56% | 58% | 66% | 61% |
| nsubj (Experimental) | 66% | 60% | 67% | 71% |
| obj (Baseline) | 63% | 59% | 68% | 68% |
| obj (Experimental) | 63% | 71% | 73% | 66% |
| obl (Baseline) | 68% | 73% | 69% | 84% |
| obl (Experimental) | 74% | 82% | 75% | 84% |

training and dev sets as training data, and evaluated on the test set. In another case we have merged the training and test sets as training data and evaluated on the dev set.

### 6.3.3 Results

We first look at the overall accuracy between the baseline and experimental models together with some results showing precision and recall for grammatical role categories relevant to verbal predicate argument structure. These are shown in Tables 21 and 22. We see an overall improvement in LAS and UAS metrics, correlated with the reduction in error rate of grammatical role discovery for argument and adjunct roles in both dev and test sets. In terms of the grammatical roles, the biggest gains seem to have been achieved in obliques, which means the system is doing a better job in identifying adjuncts and arguments. To understand whether this higher accuracy translates into better argument-adjunct distinction for the experimental model we also need to see how the syntactic attachment performed for these categories. Following the error analysis methodology we have implemented in the previous chapter, we look at whether there is a reduction in attachment errors for all the grammatical roles considered in this experiment. To do this we look at the amount of cases where the system managed to predict both the grammatical role and the syntactic attachment for arguments or adjuncts correctly and we see whether the experimental system performed better on this metric. This is the same as computing the LAS score for each of the relevant grammatical categories, which we report Table 23.

The experimental model seems to be doing better in distinguishing arguments from adjuncts. Furthermore, it manages to predict better both the grammatical function and

Table 23: Performance in Argument - Adjunct Distinction

| | | |
|---|---|---|
| nsubj (Baseline) | 1182 | 42.3% |
| nsubj (Experimental) | 1182 | 47.6% |
| obj (Baseline) | 1376 | 47.2% |
| obj (Experimental) | 1376 | 52.6% |
| obl (Baseline) | 1527 | 50.0% |
| obl (Experimental) | 1527 | 55.7% |

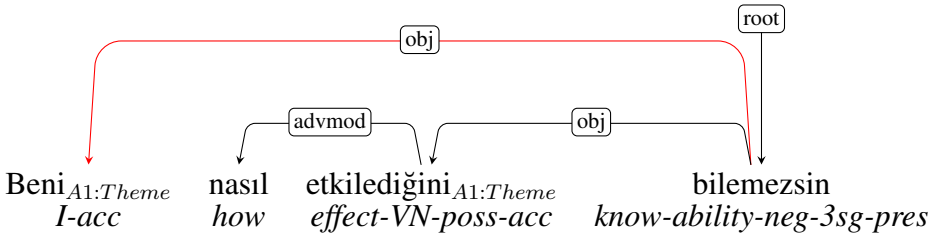


Figure 22: Dependency prediction of sentence *'Beni nasıl etkilediğini bilemezsin.'*

the syntactic head for categories relevant to verbal predicate-argument structure when using Semantic Role Labels as additional input.

### 6.4 Conclusion

In this chapter, we have used the Turkish Propbank as our dataset and have shown that a *label-first* parsing strategy that builds a dependency tree in a hierarchical set up starting from semantic role labels positively affects the UAS and LAS metrics. Specifically, confirming our hypothesis, we have seen that such a parser is able to learn the mapping between semantic roles and grammatical roles, which results in better performance in distinguishing arguments from adjuncts during parsing.

As we conclude this chapter, it is important to emphasize that the use of semantic role labels as additional input to dependency parsing does not adress the more fundemantal problems about dependency representation that were discussed in Chapter 5. In Chapter 5, we have argued that the main problem with the UD representation is its failure to decorate the lexical semantic information encoded in a predicate's meaning in the linguistic representation of lexical-functor categories. This leads a parser perform poorly in the face of sparse or unseen data, as it cannot rely on any linguistic priors to account for such data. The situation is no different when using Propbanks. As could be seen from Figure 21 in section 6.2, Propbanks are simple extensions of UD treebanks where the predicates are still not annotated with their selectional constraints such as valency and subcategorization. As a result, it is possible for the model to still make the kind of predicate-arguments structure errors that are similar to those explained in Chapter 5 even when using Propbanks. As an example,

consider the model prediction in Figure 22. Here, even though the model figures out the grammatical roles of the objects accurately, it wrongly selects two objects for the predicate *know* which only subcategorizes for one. As a result, the generated parse is in violation of the subcategorization restrictions for both the embedded predicate *etkile-* (affect) and the main predicate *bil-* (know).

All in all, it would be fair to say that using semantic role labels in Propbanks help a dependency parser to a certain degree in distinguishing arguments from adjuncts. However, as long as these semantic roles are not decorated as constraints over the verb's semantic representation, it is still possible for the parser to fail to generate a syntactic tree which obeys such requirements that emanate from the lexical semantics of verbs.

# CHAPTER 7

# LABEL FIRST PARSING WITH AN RL BASED RERANKER: A PRELIMINARY EXPERIMENT

In this final chapter of the thesis, we report the results of a preliminary experiment where the label first parser developed in Chapter 5 is combined with a *label reranker* trained using Reinforcement Learning (RL) techniques. The experimental results are not competitive against a purely supervised approach. However, we believe the experiment itself brings methodological value rather than a linguistic one due to its use of RL in dependency parsing. Therefore, the main aim of the chapter is to explain how the particular experiment was designed with the hope that it can give ideas for further research looking to integrate RL into the problem of dependency parsing.

## 7.1 Reinforcement Learning

Reinforcement Learning [95] is a family of algorithms which represent the machine learning problem at hand as an agent learning to make better decisions in an environment based on some sort of reward feedback. RL algorithms are mainly developed for settings where supervised learning with a set of gold labels is not possible and the model should learn with a trial and error process, trying to maximize better actions which lead to better results during these trials. However, hybrid approaches which combine supervised learning with reinforcement learning techniques have also been abundant in many application domains.

An RL problem is formalized as a Markov Decision Process (MDP), which is represented as a tuple <S, A, R, $\gamma$> that consists of a *state* S, a set of *actions* A, a *reward* function R, and a *discount factor* $\gamma$. The meaning of all these expressions is determined based on the exact problem one is dealing with. In a grid-world problem, the state might represent the <x, y> coordinates of the agent's location while actions would be which direction the agent will move next. In a more complicated problem such as learning to drive, state can include a variety of signals such as the position and the speed of the car, the traffic signs, position of neighboring cars and so on.

The reward function represents a feedback from the environment to the agent, signalling that its action (or action sequences) have positive or negative results. The exact specification of the reward function is usually determined by the designer of the agent.

Given this set up, the goal of an RL agent is to learn a policy ($\pi$) that maximizes rewards. As is shown in equation 14, $\pi$ is a function of the states (S) and actions (A). The agent's objective is to learn correct set of actions in each state that lead to higher overall cumulative reward. In other words, the agent's aim is to learn the best policy that maximizes rewards.

$$\pi(a|s) = p(A = a|S = s) \tag{13}$$

The algorithms by which an RL agent tries to learn the best policy that returns maximum rewards are called *policy optimization* algorithms. At the heart of the policy optimization algorithms is the Bellman Equation, which defines a so-called Q function (or state-value function) as follows.

$$Q(s, a) = r(s, a) + \gamma max_{(a')}Q(s, a) \tag{14}$$

The Q function is a definition of the value of taking an action *a* in state *s*. It states that this value is equal to the immediate reward you get from taking this action and moving to next state (s') and the cumulative discounted rewards that you gain when you follow your policy afterwards. The discount value ($\gamma$) can be fine tuned depending on how much weight one wants to give to future rewards. For example, in a chess game the reward is not based on individual actions but on the overall sequence of actions that lead to winning or losing the game. Therefore, in such an environment the discount factor can be kept high (e.g. 0.9) to maximize longer term gains. On the other hand, in an environment where each action might be immediately and independently rewarded, the discount factor can be kept minimal or be dispensed with completely.

As their name suggests, the so called Q-learning algorithms (Watkins and Dayan, 1992)[96] aim to learn Q values, i.e. the value of taking each action in each state. Accordingly, their neural versions, the Deep Q Learning (DQN) models [97], aim to approximate the Q-value for each state-action pair based on data.

### 7.2 RL in NLP

Recent years have seen an increase in the amount of works that applied RL tecniques to NLP problems. However, their application to parsing in general, and dependency parsing in particular, has been very few to non-existent. In a recent experiment that is similar to the one that we present here, Pan et al (2018)[98] tried to improve the state of the art Stanford parser using the DQN algorithm, but were unable to report promising results. On the semantic parsing side, Naseem et al (2019)[99] applied RL techniques on transition based AMR parsing and managed to report competitive results. For a detailed survey of the application of RL on NLP domains other than dependency parsing, the reader is refered to a recent comprehensive survey carried out by Luketina et al. (2019) [100].

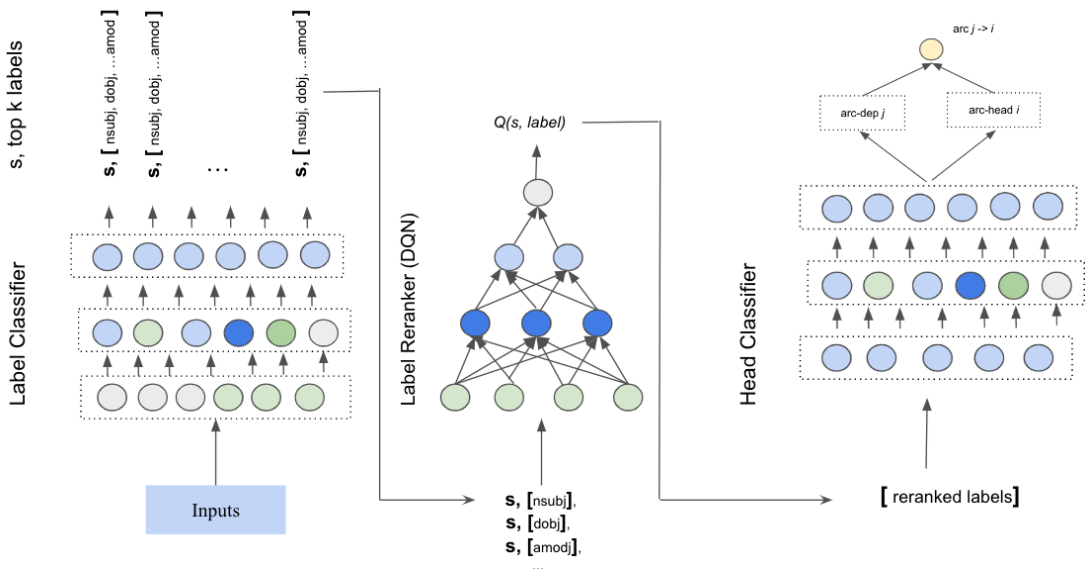


Figure 23: Information Flow in the Label Reranking Model

## 7.3 Label First Parsing using an RL Reranker

In Chapter 5, we have seen that a parser that makes better decisions in determining dependency labels performs better in resolving syntactic dependencies. Specifically, we have seen as a result of our comparative evaluation that languages that perform better on dependency labeling also perform better on syntactic parsing.

Given that bad dependency label decisions lead to wrong attachment decisions for a parser, one idea can be to design an agent which learns to recover from bad labeling decisions based on how they effect attachment. This agent operates in an environment where it observes the correlation between label predictions and attachment accuracy in the context of different sentences and tries to learn which dependency labels lead to higher attachment accuracy in different contexts (or to use the RL term, states). We designed this agent in the form of a label reranker and trained it as a DQN using the process explained in the next section.

### 7.3.1 An RL based label reranker

Our system can best be described by Figure 23 which represents the information flow within the model at inference time. We have two stand-alone LSTM models, one responsible for label prediction, the other for head-selection. In the middle of these two models is the label-reranker, which is a multi-layered feed-forward neural network (FFNN) that takes as input a <state, label> tuple and outputs the score $Q(state, label)$ for the tuple. At inference time, first the label prediction model outputs a set of *top_k* label predictions together with a state vector, *S*, representing the state for each token.

Table 24: Featureset used in state representation S for token $w_i$

| Word Features ($w_i$) | Neighbour features (for each neighbour $w_j$) |
|---|---|
| pos($w_i$) | pos($w_i$) + pos($w_j$) |
| lemma($w_i$) | lemma($w_i$)+lemma($w_j$) |
| morphology($w_i$) | morphology($w_i$) + morphology($w_j$) |
| word-embedding($w_i$) | word-embedding($w_i$) + word-embedding($w_j$) |
| | distance(($w_j$)) |
| | dependency-label($w_j$) |

The state representation is based on the linguistic properties of the token as well as the surrounding tokens (explained below). Each label in the *top_k* labels is concatenated with the state representation and these <state, label> tuples are then passed over to the label reranker for scoring. The topmost label as a result of this scoring is passed over to the head-classifier. The label reranker's scoring function is trained by observing which labels coming from the label classifier leads to higher rewards for a dependency parser, where *reward* is defined based on both label accuracy and attachment accuracy. Therefore, the aim of this procedure is for the label ranker to try to surface label predictions that lead to better attachment decisions.

During training we have chosen the *k* value in *top_k* to be 5, meaning we have passed over 5 top_k label outputs from the label network to the DQN network. This was based on the observation that in 98% of the cases, the correct label was amongst the top 5 labels produced by the label network[1].

### 7.3.2 Training the label reranker

As was mentioned in the previous section, the label reranker is trained using <state, label> pairs as inputs, based on a reward function. We describe how each of these are designed below.

*State Representation*

For each token $w_i$, the State function $s(w_i)$ extracts <key:value> pairs for $w_i$ representing its linguistic and contextual properties as in Table 24. The state representation for a token is equal to the features extracted for that token such that $s(w_i)$ := *featureset*($s(w_i)$), based on the featureset definition in the aforementioned table. While extracting "neighbour features", we keep a window size 3, meaning we look at the linguistic properties of the 3 tokens to the left and 3 the right of $w_i$.

*Training Details*

We train the DQN network based on the outputs of the label classifier and the decisions made by the head classifier. The label classifier and the head classifier are

[1] For comparison, if we had set up k to be 3, this ratio would have been dropped to 93%.

Table 25: Hyperparameters used in training label reranker

| Param | Value |
|---|---|
| Learning rate | 0.1 |
| Optimizer | Adagrad |
| MLP Depth | 8 |
| Loss Function | Pairwise Hinge Loss |
| $\gamma$ | 0 |

first trained independently and disjointly from one another using the training data. Afterwards, using the held-out dev set, we extract datapoints for the label reranker by asking the label classifier to generate *top_k* labels for each token and asking the head classifier to determine dependencies based on the generated labels. We use these datapoints to train the label reranker. As a result of this procedure, we aim to teach the label reranker to rank the *top_k* labels that lead to higher rewards, where reward is determined based on accuracy of the label prediction as well as the head prediction.

The DQN network is a FFNN which consists of 7 ReLU layers and one output layer. For optimization we have used the *Adagrad* optimizer, with the loss function as Pairwise Hinge Loss. Hinge loss carries out maximum margin classification and is therefore a commonly used loss function in listwise ranking problems where one tries to maximize the difference between a highly rated item and a lowly rated one in a list of items. The specific DQN training parameters are listed in Table 25.

*Reward Design*

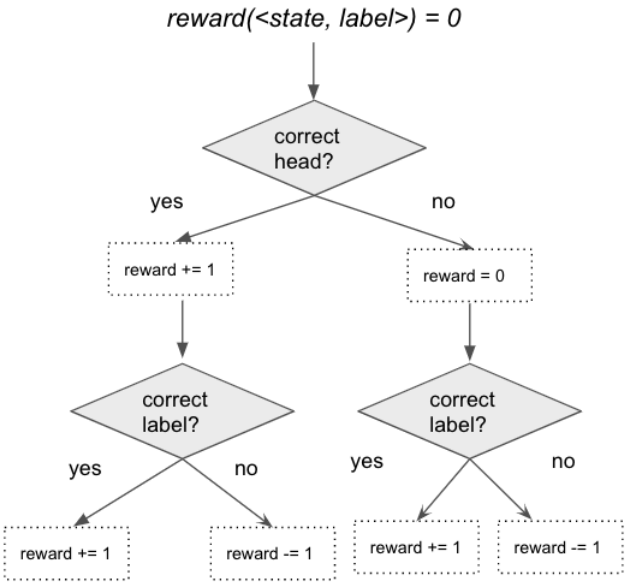


Figure 24: Reward Function Representation

We have designed a simple reward function which gives higher rewards to a <state, label> tuple when both the label is selected correctly and the head is predicted accurately for a token. Further details of the reward function can be seen in Figure 24.

This reward function gives the <state, label> inputs which lead to both wrong dependency classification and wrong label classification the lowest score (-1) and the ones that lead to accurate classification on both tasks the highest reward (+2). Cases where only the labels are accurately predicted would be rewarded as (+1) and cases where the dependency labels are wrong while the heads are classified correctly would be neutral (0).

As can be observed from Table 25, while training we have set the discount factor $\gamma$ to be 0. This means that each token is rewarded independently from one another. In other words, say in a sequence of three tokens ($w_i, w_j, w_k$), the reward from the later tokens in the sequence (e.g. $w_k$) does not propagate back and add to the cumulative reward for earlier states (e.g. $w_i$). This choice is made based on the intuition that in a sentence, the correct dependency label of a token in position $w_i$ does not have a strong impact on the prediction of the correct dependency label in position $w_j$, contrary to, for example, a PoS tagging task. However, we acknowledge that this intuition is open to debate[2] and different options for $\gamma$ should be experimented with.

### 7.4 Results and Discussion

Before reporting the results, it would be interesting to see an example where the label-reranker managed to recover the correct label out of the *top_k* labels where the correct label was lower ranked by the label network.

Consider a sentence like 2, which is taken from our eval set. We show in Figure 25 an error that was made by the LSTM model which does not use the label reranker. Here the main problem is that the label for the *root* token is predicted wrongly (*compound* rather than *root*). These kind of *root* identification errors are severe for parsing systems because it means the system failed to predict the main predicate of the sentence correctly, which affects the accuracy of overall predicate-argument structure prediction. One can also realize that the tree is cyclic, so would need to go through further cycle pruning using the Eisner algorithm.

4. Alman seyircisi bile becerisi önünde hayran kaldı.
   *'Even German audience admired her skill.'*

The model with the label reranker reranks the *root* label to the top of the list and when reparsed with this new label prediction, the predicate-argument structure of all

[2] In fact, in languages where word order is verb-initial (VSO, VOS), correct identification of the verbal predicate or *root* of the sentence early in the sequence might have a positive impact on the overall label prediction accuracy of the sequence. Therefore, giving a higher value to $\gamma$ might be a more reasonable choice for those type of languages.

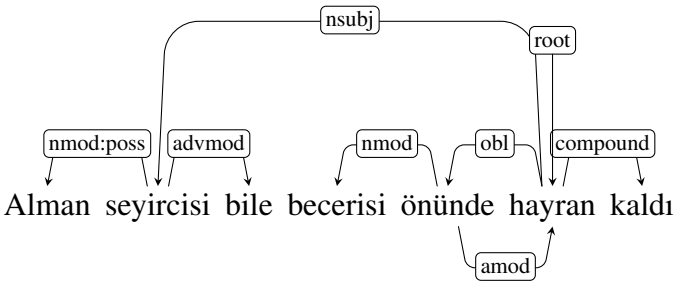


Figure 25: 'Wrong dependency tree prediction of sentence by the model without label reranker'

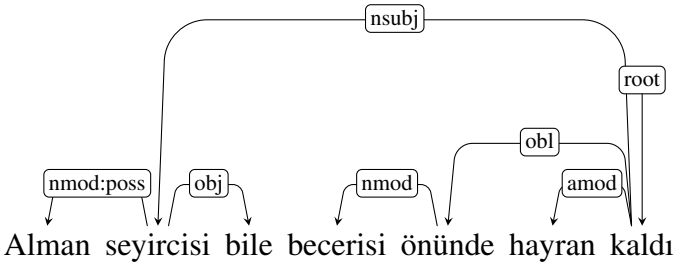


Figure 26: 'Accurate dependency tree prediction of sentence by the model with label reranker'

the tree is correctly predicted. The example in Figure 26 show the corrected parse with the reranker.

Even though examples like those in Figures 25-26 are motivating, we have found that amongst all the models developed so far for label-first dependency parsing, the reranker model achieves the lowest accuracy in relevant metrics compared to fully supervised models. This is shown in Table 26.

This means that even though in certain examples the RL reranker does a good job in correcting model errors, in more cases it fails to do so, and it even reduces model performance rather than improving it.

Table 26: Comparison of RL-based ranking model with Supervised Models

| Model | UAS | LAS |
|---|---|---|
| BERT-LSTM | 77.1 | 68.8 |
| Joint-LSTM | 74.3 | 64.9 |
| RL-Label-Reranker | 68.6 | 60.2 |

## 7.5 Conclusion

We conclude this brief chapter by stating that integrating RL based methods into dependency parsing might be a promising approach but needs to be studied further. We have seen that an RL based ranking approach might have some potential to help a parser recover from critical errors and improve its performance. However, the overall performance of the RL augmented model was lower compared to fully supervised models. We acknowledge that the experiment reported in this chapter is preliminary and there can be important design or modeling choices which need to be thought more carefully. However, the lack of previous studies in this area makes it hard to build new studies on a solid foundation and leverage previous expertise while implementing new designs. Therefore, we hope that this experiment helps further research to expand and explore more ideas on how RL can be applied to the dependency parsing problem.

# CHAPTER 8

# CONCLUSION

Predicate-argument structure lies at the heart of syntax-semantics interface of natural language. Following Clark et al (2000)[14], it can be defined as "the dependencies that hold between words with lexical functor categories and their arguments" [101]. As we have reviewed in Chapter 2, various linguistic approaches have tried to account for this interface by means of different theoretical mechanisms. Aside from radically constructionist approaches presented in Construction Grammars [26], the common agreement in all the approaches is that predicate argument structure is determined partially by verb meaning, which impose lexical semantic constraints as well as subcategorization requirements onto syntactic derivation.

This thesis presented an investigation into inducing predicate-argument structure from dependency treebanks, which aim to represent predicate-argument structure of a sentence in the form of dependency graphs where predicates are attached to their arguments by means of uni-directed and labeled dependency arcs.

The experiments that are reported in this thesis, especially in Chapter 5, have implications mainly for the *learnability* of predicate argument structure using a dependency representation. First, the purely binarized, non-compositional, and unconstrained nature of dependency formalism presented in the UD framework hinders the ability to encode grammatical restrictions that predicates project onto syntax in the dependency data. The untyped nature of lexical categories in UD results in computational models which suffer mostly in languages where arguments of a predicate can be covert in the surface form but are represented at the level of lexical conceptual structure as part of the predicate's subcategorization frame. Clear examples of this appeared in chapter 5 when we compared the variation in parsing performance in different types of languages. In non-analytical languages like Turkish, a recurrent problem was being able to induce the correct dependencies between predicates and arguments in sentences with multiple predicates where one or more of the arguments of a predicate could be covert in the surface representation.

The above finding means that an effort that aims to induce predicate argument structure from data, especially for languages like Turkish, needs to employ a more elaborate representation where syntax-semantics correspondance is modelled more transparently in the surface form, similar to the Montague approach to syntax semantics interface, an implementation of which can be found in compositional approaches like CCGs which were reviewed in Chapter 2. Specifically, to be able to learn how to

associate the correct arguments with correct predicates, the need is to type semantic constraints such as valency over the meaning bearing units (which can be morphological or lexical) that project arguments in the syntax. However, the lack of such semantic notions as valency in a dependency formalism does not make this possible.

In languages like Turkish, morphological meaning bearing units introduce their own semantic constraints on predicate argument structure by means of which they can modify the subcategorization frame of a predicate by adding arguments to it or deleting arguments from it. This implies that the linguistic representation from which one induces predicate argument structure for a language like Turkish should also accurately encode the linguistic properties of these units, and account for their type-changing role in syntax. However, the representation that UD provides which casts all the langauges (whether morphologically complex or not) into a unified representation further impedes efforts that try to learn predicate argument structure from data in this framework.

All in all, the main conclusion of this thesis is that employing a dependency representation as the one provided in UD for learning predicate argument structure only works well for languages that are morphologically simple and analytic, which, due to lack of rich morphological inflection and derivation, represent predicates and arguments overtly in the surface form.

## 8.1 Future Work

We believe that the label first parsing methodology and the experiments with this methodology in Chapter 5 lay the ground for future work as described below.

**Unified Multilingual parsing**. In chapter 5, we have fine-tuned BERT models for 7 different languages separately on the dependency label prediction task. We have seen that this has led to improvements on certain languages in terms of overall parsing accuracy while not in others. We have also shown that there is a strong correlation between dependency label accuracy and attachment accuracy in all languages considered.

One way to further extend this experiment is to train a unified, multilingual dependency labeling model using BERT which can classify dependency labels for multiple languages at once. One challenge of this approach is the difference in dependency tagsets used in each language. While certain languages like Russian and Finnish use a more extensive tagset, languages like Korean and Chinese use relatively fewer tags. To avoid data sparsity that might be introduced due to the divergence in tagsets in a unified model, one might consider grouping languages with similar tagsets together in such an experiment.

**Joint BERT based label first parsing**. In the model proposed in Chapter 5, the BERT based dependency labeler and the LSTM based dependency head classifier were trained disjointly from one another. A logical next step is to train a completely

joint model where the BERT model and the LSTM model is optimized together. This would lead to creation of trainable dependencies between the two architectures and more importantly allow the losses from the LSTM model to backpropagate to the BERT model. The hypothesis is that if such a model can be properly optimized, this joint training would further improve the accuracy of both of the tasks due to the strong correlation between them.

**Joint SRL and Dependency Parsing**. In Chapter 6, we have made the assumption that early access to semantic role labels in a label-first parsing setting should improve a) identification of the correct argument roles and distinguishing arguments from adjuncts and in turn b) the overall parsing accuracy. The results presented in the chapter have confirmed our hypothesis, showing improvements on both grounds. As we have explained, the experiment did not jointly optimize for semantic role labels during training. Therefore, further research should be implemented which shows whether a hierarchical, multilayered set up which starts parsing from semantic role classification, followed by dependency label classification leading up to dependency tree construction can help improve all three tasks at the same time compared to current approaches to joint semantic and syntactic parsing which mostly predict semantic roles over an already built dependency tree.

## REFERENCES


[1] D. Marcheggiani, A. Frolov, and I. Titov, "A simple and accurate syntax-agnostic neural model for dependency-based semantic role labeling," *arXiv preprint arXiv:1701.02593*, 2017.

[2] Y. Zhang, P. Qi, and C. D. Manning, "Graph convolution over pruned dependency trees improves relation extraction," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, (Brussels, Belgium), pp. 2205–2215, Association for Computational Linguistics, Oct.-Nov. 2018.

[3] K. Chen, R. Wang, M. Utiyama, L. Liu, A. Tamura, E. Sumita, and T. Zhao, "Neural machine translation with source dependency representation," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, (Copenhagen, Denmark), pp. 2846–2852, Association for Computational Linguistics, Sept. 2017.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, (Minneapolis, Minnesota), pp. 4171–4186, Association for Computational Linguistics, June 2019.

[5] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, *et al.*, "Palm: Scaling language modeling with pathways," *arXiv preprint arXiv:2204.02311*, 2022.

[6] D. Kondratyuk and M. Straka, "75 languages, 1 model: Parsing universal dependencies universally," *arXiv preprint arXiv:1904.02099*, 2019.

[7] M.-C. de Marneffe, C. D. Manning, J. Nivre, and D. Zeman, "Universal Dependencies," *Computational Linguistics*, vol. 47, pp. 255–308, June 2021.

[8] K. Oflazer, "Turkish and its challenges for language processing," vol. 48, p. 639–653, dec 2014.

[9] U. Sulubacak, M. Gokirmak, F. Tyers, Ç. Çöltekin, J. Nivre, and G. Eryiğit, "Universal Dependencies for Turkish," in *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, (Osaka, Japan), pp. 3444–3454, The COLING 2016 Organizing Committee, Dec. 2016.

[10] U. Türk, 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.

[11] Ç. Çöltekin, "A grammar-book treebank of turkish," in *International Workshop on Treebanks and Linguistic Theories (TLT14)*, p. 35.

[12] T. Kayadelen, A. Öztürel, and B. Bohnet, "A gold standard dependency treebank for turkish," in *Proceedings of the 12th language resources and evaluation conference*, pp. 5156–5163, 2020.

[13] D. Zeman, M. Popel, M. Straka, J. Hajič, J. Nivre, F. Ginter, J. Luotolahti, S. Pyysalo, S. Petrov, M. Potthast, F. Tyers, E. Badmaeva, M. Gokirmak, A. Nedoluzhko, S. Cinková, J. Hajič jr., J. Hlaváčová, V. Kettnerová, Z. Urešová, J. Kanerva, S. Ojala, A. Missilä, C. D. Manning, S. Schuster, S. Reddy, D. Taji, N. Habash, H. Leung, M.-C. de Marneffe, M. Sanguinetti, M. Simi, H. Kanayama, V. de Paiva, K. Droganova, H. Martínez Alonso, Ç. Çöltekin, U. Sulubacak, H. Uszkoreit, V. Macketanz, A. Burchardt, K. Harris, K. Marheinecke, G. Rehm, T. Kayadelen, M. Attia, A. Elkahky, Z. Yu, E. Pitler, S. Lertpradit, M. Mandl, J. Kirchner, H. F. Alcalde, J. Strnadová, E. Banerjee, R. Manurung, A. Stella, A. Shimada, S. Kwak, G. Mendonça, T. Lando, R. Nitisaroj, and J. Li, "CoNLL 2017 shared task: Multilingual parsing from raw text to Universal Dependencies," in *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, (Vancouver, Canada), pp. 1–19, Association for Computational Linguistics, Aug. 2017.

[14] S. Clark, J. Hockenmaier, and M. Steedman, "Building deep dependency structures with a wide-coverage ccg parser," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, (USA), p. 327–334, Association for Computational Linguistics, 2002.

[15] L. Talmy, "Lexicalization patterns: Semantic structure in lexical forms," *Grammatical categories and the lexicon*, 1985.

[16] L. Talmy, "Path to realization: A typology of event conflation," 1991.

[17] N. Chomsky, *Lectures on Government and Binding*. Dordrecht: Foris., 1981.

[18] G. Ramchand, *Verb Meaning and the Lexicon: A First-phase Syntax*. Cambridge Studies in Linguistics, Cambridge University Press, 2009.

[19] S. Pinker, *Learnability and Cognition: The Acquisition of Argument Structure*. Bradford books, MIT Press, 1989.

[20] N. Chomsky, *Remarks on Nominalization*. Linguistics Club, Indiana University, 1968.

[21] C. Fillmore, *The Grammar of 'hitting' and 'breaking'*, pp. 120–133. Ginn, 1970.

[22] B. Levin, *English Verb Classes and Alternations: A Preliminary Investigation*. University of Chicago Press, 1993.

[23] B. Levin and M. Rappaport Hovav, *Argument Realization*. Research Surveys in Linguistics, Cambridge University Press, 2005.

[24] D. Dowty, "Thematic proto-roles and argument selection," *Language*, vol. 67, no. 3, pp. 547–619, 1991.

[25] A. Goldberg, *Constructions: A Construction Grammar Approach to Argument Structure*. Cognitive Theory of Language and Culture Series, University of Chicago Press, 1995.

[26] L. Michaelis, "Construction grammar," *syntax and semantics*, vol. 17, pp. 243–262, 2006.

[27] A. E. Goldberg, "Constructions: A new theoretical approach to language," *Trends in cognitive sciences*, vol. 7, no. 5, pp. 219–224, 2003.

[28] M. Steedman, *Surface Structure and Interpretation*. Linguistic Inquiry Series, MIT Press, 1996.

[29] M. Steedman, "The syntactic process," 2000.

[30] K. Ajdukiewicz, "Die syntaktische konnexitat," *Studia philosophica*, pp. 1–27, 1935.

[31] Y. Bar-Hillel, "A quasi-arithmetical notation for syntactic description," *Language*, vol. 29, no. 1, pp. 47–58, 1953.

[32] M. McConville, *An Inheritance Based Hierarchical Lexicon in Combinatory Categorial Grammar*. PhD thesis, University of Edinburgh, 2007.

[33] M. Steedman, "Combinatory categorial grammar," in *Current Approaches to Syntax*, pp. 389–420, De Gruyter Mouton.

[34] J. Gruber, "Lexical structures in syntax and semantics," 1976.

[35] R. Jackendoff, *Semantic Interpretation in Generative Grammar*. Studies in linguistics series, MIT Press, 1974.

[36] S. Kubler, R. McDonald, and J. Nivre, *Dependency Parsing*. Morgan & Claypool Publishers, 2009.

[37] M. Kuhlmann and J. Nivre, "Mildly non-projective dependency structures," in *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, (Sydney, Australia), pp. 507–514, Association for Computational Linguistics, July 2006.

[38] M. Collins, J. Hajic, L. Ramshaw, and C. Tillmann, "A statistical parser for Czech," in *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, (College Park, Maryland, USA), pp. 505–512, Association for Computational Linguistics, June 1999.

[39] H. Yamada and Y. Matsumoto, "Statistical dependency analysis with support vector machines," in *Proceedings of the Eighth International Conference on Parsing Technologies*, (Nancy, France), pp. 195–206, Apr. 2003.

[40] M. Collins, "Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms," in *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pp. 1–8, Association for Computational Linguistics, July 2002.

[41] R. McDonald and J. Nivre, "Characterizing the errors of data-driven dependency parsing models," *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, vol. 112, no. 1, p. 155, 2007.

[42] J. Nivre, "Algorithms for deterministic incremental dependency parsing," *Computational Linguistics*, vol. 34, no. 4, pp. 513–553, 2008.

[43] J. Nivre, "An efficient algorithm for projective dependency parsing," in *Proceedings of the Eighth International Conference on Parsing Technologies*, (Nancy, France), pp. 149–160, Apr. 2003.

[44] J. Nivre, "Incrementality in deterministic dependency parsing," in *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, (Barcelona, Spain), pp. 50–57, Association for Computational Linguistics, July 2004.

[45] J. Nivre, "Non-projective dependency parsing in expected linear time," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, (Suntec, Singapore), pp. 351–359, Association for Computational Linguistics, Aug. 2009.

[46] P. Qi and C. D. Manning, "Arc-swift: A novel transition system for dependency parsing," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, (Vancouver, Canada), pp. 110–117, Association for Computational Linguistics, July 2017.

[47] R. McDonald, F. Pereira, K. Ribarov, and J. Hajič, "Non-projective dependency parsing using spanning tree algorithms," in *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, (Vancouver, British Columbia, Canada), pp. 523–530, Association for Computational Linguistics, Oct. 2005.

[48] R. McDonald and F. Pereira, "Online learning of approximate dependency parsing algorithms," in *11th Conference of the European Chapter of the Association for Computational Linguistics*, (Trento, Italy), pp. 81–88, Association for Computational Linguistics, Apr. 2006.

[49] Y.-J. Chu and T.-H. Liu, "On the shortest arborescence of a directed graph," *Science Sinica*, vol. 14, pp. 1396–1400, 1965.

[50] J. Edmonds, "Optimum branchings," *Journal of Research of the National Bureau of Standards*, vol. 71b, pp. 233–240, 1967.

[51] T. Dozat and C. D. Manning, "Deep biaffine attention for neural dependency parsing," *arXiv preprint arXiv:1611.01734*, 2016.

[52] T. Dozat, P. Qi, and C. D. Manning, "Stanford's graph-based neural dependency parser at the CoNLL 2017 shared task," in *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, (Vancouver, Canada), pp. 20–30, Association for Computational Linguistics, Aug. 2017.

[53] T. Dozat, *Arc-Factored Biaffine Dependency Parsing*. PhD thesis, Stanford University, California, US, 2019.

[54] E. Kiperwasser 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.

[55] K. Hashimoto, C. Xiong, Y. Tsuruoka, and R. Socher, "A joint many-task model: Growing a neural network for multiple NLP tasks," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, (Copenhagen, Denmark), pp. 1923–1933, Association for Computational Linguistics, Sept. 2017.

[56] X. Zhang, J. Cheng, and M. Lapata, "Dependency parsing as head selection," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, (Valencia, Spain), pp. 665–676, Association for Computational Linguistics, Apr. 2017.

[57] R. Jackendoff, *X' syntax: A study of phrase structure*. MIT, 1977.

[58] A. Przepiórkowski and A. Patejuk, "Arguments and adjuncts in Universal Dependencies," in *Proceedings of the 27th International Conference on Computational Linguistics*, (Santa Fe, New Mexico, USA), pp. 3837–3852, Association for Computational Linguistics, Aug. 2018.

[59] R. Kaplan, "Lexical functional grammar, a formal system for grammatical representation. the mental representation of grammatical relations, ed. by joan bresnan, 173–281," 1982.

[60] J. Bresnan, A. Asudeh, I. Toivonen, and S. Wechsler, *Lexical-functional syntax*. John Wiley & Sons, 2015.

[61] K. Hale and S. Keyser, *Prolegomenon to a Theory of Argument Structure*. Linguistic Inquiry Monographs, MIT Press, 2002.

[62] W. Croft, *Syntactic categories and grammatical relations: The cognitive organization of information*. University of Chicago Press, 1991.

[63] Z. Li, J. Cai, S. He, and H. Zhao, "Seq2seq dependency parsing," in *Proceedings of the 27th International Conference on Computational Linguistics*, (Santa Fe, New Mexico, USA), pp. 3203–3214, Association for Computational Linguistics, Aug. 2018.

[64] D. johanka Spoustová and M. Spousta, "Dependency parsing as a sequence labeling task," in *Prague Bulletin of Mathematical Linguistics*, 2010.

[65] E. Kiperwasser and M. Ballesteros, "Scheduled multi-task learning: From syntax to translation," *Transactions of the Association for Computational Linguistics*, vol. 6, pp. 225–240, 2018.

[66] M. Strzyz, D. Vilares, and C. Gómez-Rodríguez, "Viable dependency parsing as sequence labeling," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, (Minneapolis, Minnesota), pp. 717–723, Association for Computational Linguistics, June 2019.

[67] J. Nivre, Ž. Agić, L. Ahrenberg, L. Antonsen, M. J. Aranzabe, M. Asahara, L. Ateyah, M. Attia, A. Atutxa, E. Badmaeva, M. Ballesteros, E. Banerjee, S. Bank, J. Bauer, K. Bengoetxea, R. A. Bhat, E. Bick, C. Bosco, G. Bouma, S. Bowman, A. Burchardt, M. Candito, G. Caron, G. Cebiroğlu Eryiğit, G. G. A. Celano, S. Cetin, F. Chalub, J. Choi, Y. Cho, S. Cinková, Ç. Çöltekin, M. Connor, M.-C. de Marneffe, V. de Paiva, A. Diaz de Ilarraza, K. Dobrovoljc, T. Dozat, K. Droganova, M. Eli, A. Elkahky, T. Erjavec, R. Farkas, H. Fernandez Alcalde, J. Foster, C. Freitas, K. Gajdošová, D. Galbraith, M. Garcia, F. Ginter, I. Goenaga, K. Gojenola, M. Gökırmak, Y. Goldberg, X. Gómez Guinovart, B. Gonzáles Saavedra, M. Grioni, N. Grūzītis, B. Guillaume, N. Habash, J. Hajič, J. Hajič jr., L. Hà Mỹ, K. Harris, D. Haug, B. Hladká, J. Hlaváčová, P. Hohle, R. Ion, E. Irimia, A. Johannsen, F. Jørgensen, H. Kaşıkara, H. Kanayama, J. Kanerva, T. Kayadelen, V. Kettnerová, J. Kirchner, N. Kotsyba, S. Krek, S. Kwak, V. Laippala, L. Lambertino, T. Lando, P. Lê Hồng, A. Lenci, S. Lertpradit, H. Leung, C. Y. Li, J. Li, N. Ljubešić, O. Loginova, O. Lyashevskaya, T. Lynn, V. Macketanz, A. Makazhanov, M. Mandl, C. Manning, R. Manurung, C. Mărănduc, D. Mareček, K. Marheinecke, H. Martínez Alonso, A. Martins, J. Mašek, Y. Matsumoto, R. McDonald, G. Mendonça, A. Missilä, V. Mititelu, Y. Miyao, S. Montemagni, A. More, L. Moreno Romero, S. Mori, B. Moskalevskyi, K. Muischnek, N. Mustafina, K. Müürisep, P. Nainwani, A. Nedoluzhko, L. Nguyễn Thị, H. Nguyễn Thị Minh, V. Nikolaev, R. Nitisaroj, H. Nurmi, S. Ojala, P. Osenova, L. Øvrelid, E. Pascual, M. Passarotti, C.-A. Perez, G. Perrier, S. Petrov, J. Piitulainen, E. Pitler, B. Plank, M. Popel, L. Pretkalniņa, P. Prokopidis, T. Puolakainen, S. Pyysalo, A. Rademaker, L. Real, S. Reddy, G. Rehm, L. Rinaldi, L. Rituma, R. Rosa, D. Rovati, S. Saleh, M. Sanguinetti, B. Saulīte, Y. Sawanakunanon, S. Schuster, D. Seddah, W. Seeker, M. Seraji, L. Shakurova, M. Shen, A. Shimada, M. Shohibussirri, N. Silveira, M. Simi, R. Simionescu, K. Simkó, M. Šimková, K. Simov, A. Smith, A. Stella, J. Strnadová, A. Suhr, U. Sulubacak, Z. Szántó, D. Taji, T. Tanaka, T. Trosterud,

A. Trukhina, R. Tsarfaty, F. Tyers, S. Uematsu, Z. Urešová, L. Uria, H. Uszkoreit, G. van Noord, V. Varga, V. Vincze, J. N. Washington, Z. Yu, Z. Žabokrtský, D. Zeman, and H. Zhu, "Universal dependencies 2.0 – CoNLL 2017 shared task development and test data," 2017. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

[68] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *ICLR*, 2013.

[69] U. Türk, F. Atmaca, Ş. B. Özateş, G. Berk, S. T. Bedir, A. Köksal, B. Ö. Başaran, T. Güngör, and A. Özgür, "Boun treebank," 2022.

[70] M. Straka, J. Hajič, and J. Straková, "UDPipe: Trainable pipeline for processing CoNLL-U files performing tokenization, morphological analysis, POS tagging and parsing," in *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, (Portorož, Slovenia), pp. 4290–4297, European Language Resources Association (ELRA), May 2016.

[71] Şaziye B. Özateş, *Deep Learning Based Dependency Parsing for Turkish*. PhD thesis, Boğaziçi Univesity, Istanbul, Turkey, 2022.

[72] C. Alberti, K. Lee, and M. Collins, "A bert baseline for the natural questions," *ArXiv*, vol. abs/1901.08634, 2019.

[73] Y. Liu, "Fine-tune bert for extractive summarization," *arXiv preprint arXiv:1903.10318*, 2019.

[74] J. Zhu, Y. Xia, L. Wu, D. He, T. Qin, W. Zhou, H. Li, and T.-Y. Liu, "Incorporating bert into neural machine translation," *arXiv preprint arXiv:2002.06823*, 2020.

[75] Y. Goldberg, "Assessing bert's syntactic abilities," *arXiv preprint arXiv:1901.05287*, 2019.

[76] G. Jawahar, B. Sagot, and D. Seddah, "What does BERT learn about the structure of language?," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, (Florence, Italy), pp. 3651–3657, Association for Computational Linguistics, July 2019.

[77] J. Hewitt and C. D. Manning, "A structural probe for finding syntax in word representations," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, (Minneapolis, Minnesota), pp. 4129–4138, Association for Computational Linguistics, June 2019.

[78] X. Song, A. Salcianu, Y. Song, D. Dopson, and D. Zhou, "Fast wordpiece tokenization," *arXiv preprint arXiv:2012.15524*, 2020.

[79] R. Tsarfaty, 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," in *Proceedings of the NAACL HLT 2010 First Workshop on Statistical Parsing of Morphologically-Rich Languages*, (Los Angeles, CA, USA), pp. 1–12, Association for Computational Linguistics, June 2010.

[80] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[81] D. Zeman, 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," in *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, (Brussels, Belgium), pp. 1–21, Association for Computational Linguistics, Oct. 2018.

[82] B. Can, H. Aleçakır, S. Manandhar, and C. Bozşahin, "Joint learning of morphology and syntax with cross-level contextual information flow," *Natural Language Engineering*, vol. 28, no. 6, pp. 763–795, 2022.

[83] P. Kingsbury and M. Palmer, "From TreeBank to PropBank," in *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'02)*, (Las Palmas, Canary Islands - Spain), European Language Resources Association (ELRA), May 2002.

[84] M. Palmer, D. Gildea, and P. Kingsbury, "The proposition bank: An annotated corpus of semantic roles," *Comput. Linguist.*, vol. 31, p. 71–106, mar 2005.

[85] G. G. Şahin, *Building of Turkish PropBank and Semantic Role Labeling of Turkish*. PhD thesis, Istanbul Technical University, Istanbul, Turkey, 2018.

[86] C. Bonial, "English propbank annotation guidelines,"

[87] V. Punyakanok, D. Roth, and W.-t. Yih, "The importance of syntactic parsing and inference in semantic role labeling," *Comput. Linguist.*, vol. 34, p. 257–287, jun 2008.

[88] D. Gildea and M. Palmer, "The necessity of parsing for predicate argument recognition," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, (USA), p. 239–246, Association for Computational Linguistics, 2002.

[89] D. Gildea and D. Jurafsky, "Automatic labeling of semantic roles," *Computational Linguistics*, vol. 28, no. 3, pp. 245–288, 2002.

[90] W. Che, Z. Li, Y. Hu, Y. Li, B. Qin, T. Liu, and S. Li, "A cascaded syntactic and semantic dependency parsing system," in *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*, (Manchester, England), pp. 238–242, Coling 2008 Organizing Committee, Aug. 2008.

[91] R. Johansson and P. Nugues, "Dependency-based syntactic–semantic analysis with PropBank and NomBank," in *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*, (Manchester, England), pp. 183–187, Coling 2008 Organizing Committee, Aug. 2008.

[92] G. Musillo and P. Merlo, "Accurate parsing of the Proposition Bank," in *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, (New York City, USA), pp. 101–104, Association for Computational Linguistics, June 2006.

[93] S.-t. Yi and M. Palmer, "The integration of syntactic parsing and semantic role labeling," in *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, (Ann Arbor, Michigan), pp. 237–240, Association for Computational Linguistics, June 2005.

[94] T. Shi, I. Malioutov, and O. Irsoy, "Semantic role labeling as syntactic dependency parsing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Online), pp. 7551–7571, Association for Computational Linguistics, Nov. 2020.

[95] R. Sutton and A. Barto, "Reinforcement learning: An introduction," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 1054–1054, 1998.

[96] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, pp. 279–292, 1992.

[97] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[98] C. Pan, S. T. Barratt, and J. T. Barratt, "Improving the neural dependency parser," 2018.

[99] T. Naseem, A. Shah, H. Wan, R. Florian, S. Roukos, and M. Ballesteros, "Rewarding Smatch: Transition-based AMR parsing with reinforcement learning," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, (Florence, Italy), pp. 4586–4592, Association for Computational Linguistics, July 2019.

[100] J. Luketina, N. Nardelli, G. Farquhar, J. Foerster, J. Andreas, E. Grefenstette, S. Whiteson, and T. Rocktäschel, "A survey of reinforcement learning informed by natural language," *arXiv preprint arXiv:1906.03926*, 2019.

[101] J. Hockenmaier, "Parsing with generative models of predicate-argument structure," in *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, (USA), p. 359–366, Association for Computational Linguistics, 2003.

## CURRICULUM VITAE

### PERSONAL INFORMATION

**Surname, Name**: Kayadelen, Tolga

**E-mail**: kayadelen.tolga@gmail.com

### EDUCATION

| Degree | Department | Institution | Year of Graduation |
|---|---|---|---|
| M.A. | English Linguistics | Mersin University | 2008 |
| B.A. | English Literature | Erciyes University | 2005 |

### PROFESSIONAL EXPERIENCE

| Year | Institution | Enrollment |
|---|---|---|
| 2016-present | Google UK | Senior Computational Linguist |
| 2014-2016 | Google Turkey | Project Manager |
| 2011-2014 | Erciyes University | Lecturer in Linguistics |
| 2005-2011 | Mersin University | Lecturer |

### PUBLICATIONS

**Kayadelen, T.**, Öztürel, A., and Bohnet, B. (2020). A gold standard dependency treebank for Turkish. *Proceedings of the 12th International Conference on Language Resources and Evaluation (LREC)*(pp. 5156-5163).

Öztürel, A., **Kayadelen, T.**, and Demirşahin, I. (2019). A syntactically expressive morphological analyzer for Turkish. *Proceedings of the 14th International Conference on Finite-State Methods and Natural Language Processing (FSMNLP)* (pp. 65-75).

Kagy, J.-F., **Kayadelen, T.**, Ma, J., Rostamizadeh, A., and Stranadova, J. (2019). The practical challenges of active learning: lessons learned from live experimentation. Presented at *2019 ICML Workshop on Human in the Loop Learning (HILL 2019)* Long Beach, USA. `https://arxiv.org/abs/1907.00038`.

**Kayadelen, T.** (2011). Bare Noun Direct Objects and Telicity in Turkish. *Mersin Üniversitesi Dil ve Edebiyat Dergisi*, 5(2).

Aksan, Y., **Kayadelen, T.**, and Yücel, Ö. (2010). Olay anlambilimi açısından Türkçe durum eylemleri üzerine kimi gözlemler. *Proceedings of the 23rd National conference on linguistics* (pp. 251-269).