# Effect Of Contextual Embeddings on Graph-Based Dependency Parsing

by

**Berkay Furkan Önder**

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Computer Science and Engineering


May 13, 2020

**Effect Of Contextual Embeddings on Graph-Based Dependency Parsing**

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

**Berkay Furkan Önder**

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Prof. Dr. Deniz Yuret

_____

Assoc. Prof. Arzucan Özgür Türkmen

_____

Prof. Dr. Engin Erzin

Date: _____

*Dedicated to my family and my friends and everyone who cares about well-being of others, especially in hard times.*

# ABSTRACT

I demonstrate the effect of contextual embeddings on transition and graph-based parsing methods and test our contribution, the structured meta biaffine decoder, using various graph-based parsing algorithms.

As Koç University Graph-Based parsing team, we implemented a graph-based dependency parsing model in order to perform syntactic and semantic analysis of given sentences. Our neural graph-based parser consists of two main parts, which are encoder and decoder. The encoder forms continuous feature vectors from provided sentences for the neural graph-based parser to process the texts properly, whereas the decoder produces the parse tree from the output of the neural parser, by first producing a graph representation of the output. We participated in CoNLL 2018 Shared Task with the parsing model we created, and had the opportunity to run our model on 61 different data sets formed with texts in 41 different languages. We took advantage of natural language processing and deep learning techniques, including graph-based dependency parsing algorithms.

# ÖZETÇE

Yaptığım çalışmada, verilen cümlelerin dil bilgisi analizini, cümledeki her bir kelimenin yapısını ve cümle içindeki görevini tespit edebilecek bir sistem geliştirdim. Bağlılık ayrıştırma adı verilen bu iş için geliştirilen sistem, temelde çizge tabanlı algoritmaları ve derin öğrenme tekniklerini kullanmaktadır. Geçiş ve çizge tabanlı yöntemler ile yapılan bağlılık ayrıştırma işleminde, kelime vektörlerine, kelime bağlamlarını eklemenin etkisini gözlemledim. Ayrıca, SMeta adını verdiğim çözücü yapısını, çizge tabanlı bağlılık ayrıştırma yöntemleri ile test ettim.

Dil bilgisi analizi için oluşturulan çizge tabanlı model, kodlayıcı ve çözücü olmak üzere iki ana bileşenden oluşmaktadır. Kodlayıcı, metinde yer alan cümleleri modelin işleyebileceği şekilde temsil edecek vektörleri oluşturur. çözücü kısım ise, her bir cümleyi temsil eden vektörleri kullanarak, cümlenin yapı ve dil bilgisi analizini gerçekleştirir ve bunun sonucu olarak cümle içerisinde yer alan kelimeler arasındaki bağlantıları temsil eden bağlılık çizgesini oluşturur.

Koç üniversitesi takımı olarak oluşturduğumuz model ile 2018 Evrensel Bağlılık Analizi yarışmasında yer aldık ve modelimizi 41 farklı dilde oluşturulmuş 61 veri seti üzerinde çalıştırma imkanı yakaladık. Modelin oluşturulma aşamasında derin öğrenme tekniklerinden ve doğal dil işleme tekniklerinden, özellikle çizge tabanlı bağlılık ayrıştırma yönteminden yararlandık.

# ACKNOWLEDGMENTS

I would like to thank my advisor, Deniz Yuret, for his guidance and support during my graduate education. I would also thank my thesis commitee members, Arzucan Özgür Türkmen and Engin Erzin for their constructive feedbacks which helped improving my thesis.

I also want to thank my collegues in Koç University Artificial Intelligence Laboratory, especially Cemil Cengiz, for his support and for sharing valuable ideas with me which definitely helped my resarch to progress. And, thanks to Can Gümeli, previous member of the laboratory, who was my partner during our project SParse. I believe we had a nice collaboration working for our project together. I want to thank Ulaş Sert for being kind and helpful in difficult situations and for making them easier. And, thanks to my friends for their trust and support when I needed. I also want to thank Scientific and Technological Research Council of Turkey (TÜBİTAK) for funding our project.

Finally, I appreciate my family for their support and patience throughout my entire master's study, including these quarantine days making our life harder while reminding us the importance of our health once again, and past members of my family for leaving valuable memories behind.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

Chapter 1

# INTRODUCTION

Dependency parsing is a way of finding and analyzing the syntax and the grammar of a language. The method has become popular in last few years because efficient models have been produced on several languages with the aid of dependency parsing methods and dependency parsing can be applied to various areas of research such as natural language understanding, human-computer interaction and machine translation.

Techniques based on dependency analysis have drawn attention for finding the grammatical structure of a sentence. Their progress is due to their ability to process several languages having distinct grammar. Various methods for analyzing the structure of a sentence have been developed after dependency parsing gained importance from researchers in the field. As an example, the Conference on Computational Natural Language Learning (CoNLL) organized a shared task in 2017 [Zeman et al., 2017], in which participants needed to develop artificial intelligence models which can learn to identify grammatical structure and relations between words of sentences in different languages. To achieve this, texts of more than fifty languages are gathered from various sources such as news, books and internet. Then, linguists determine relations between words in each sentence and the sentences with found relations are used by computer programs during learning of grammatical structure of each language. However, there has been some inconsistencies about word connections in different models because word separations are not clear in some languages.

CoNLL 2018 Shared Task [Zeman et al., 2018] further improved the approach by adding two new main methods for comparing developed models. Methods introduced

in the task pinpoint figuring out word structures and relations between words using developed models. Another contribution of CoNLL 2018 Shared Task is the inclusion of texts from new languages which were not processed during previous tasks. As the task extends the scope of dependency parsing through new languages, its difficulty is also increased for the participants since participants now need to consider new comparison metrics while developing their parsing model and process new languages using their models.

# Chapter 2

# DEPENDENCY PARSING

Dependency Parsing regards all sentences as being formed by words which are connected to each other in a hierarchical relationship. The relations between words are called "dependency relations". A dependency relation is a directed link between two words and shows that one of the words, which is called the "dependent", is related to word it is linked to, which is called the "head". Besides, each dependency relation comprises a label, which describes the type of the relation. For instance, the label "attribute" shows that the dependent word is attribute of its head. [Kübler et al., 2009] As [Kırnap, 2018] states, two words are linked to form a hierarchical relationship in a dependency relation. Areas in natural language processing, like machine translation and information retrieval benefit from these dependencies.

A model developed for dependency parsing is supposed to be able to link each word of a given sentence to the appropriate head word, and to determine correct labels for found links. By finding an appropriate head and label for each word in a sentence, the model creates a structure, which is called "dependency graph". The objective of dependency parsing is therefore to create a dependency graph from each given sentence. There are several ways to achieve this objective.

## *2.1 Data Driven Dependency Parsing*

Data-driven dependency parsing aims to build a well-designed model which can estimate the correct dependency tree for each given sentence. Data driven methods exploit machine learning techniques when trying to find the dependency graph for the sentence. An important part of data driven methods are "supervised", which assume that sentences used for training the parsing model are given with their correct

dependency graphs. It is necessary to come up with two main challenges in supervised methods. The learning problem means training a model from given sentences with their supposedly correct dependency graphs, which can be used for parsing other sentences. The other challenge is named as parsing problem. In the parsing problem, trained model needs to create the most suitable dependency graph corresponding to each given sentence.

### *2.1.1 Learning problem*

In the learning problem, we are trying to find values for the parameters which will result in the most accurate predictions of dependency trees for each sentence. [Kiperwasser and Goldberg, 2016] suggest bidirectional recurrent neural networks (BiRNNs) for describing word encodings, as opposed to representing word with attributes which are formed by hand. They use bidirectional long short-term memories (BiLSTMs) as a kind of BiRNNs in their work because the words of each sentence can be illustrated effectively using BiLSTMs, including their container sub sentences. Kiperwasser and Goldberg also state that increased number of features are provided to lower vector sizes in recent works, instead of using separate vector for each feature value. This change is due to the increase in usage of neural networks. They mostly use a chain of BiRNNs in their work. These structures are called "deep BiRNNs".

### *2.1.2 Parsing problem*

Data driven models vary on implemented parsing techniques, the ways of creating the model and parsing new sentences using that model. Transition based and graph based parsing methods constitute two of the most widely known methods of dependency parsing. While transition based parsing methods rely on [Kudo and Matsumoto, 2002] and [Yamada and Matsumoto, 2003]s greedy algorithms which are built upon stacks, [McDonald et al., 2005]s maximum spanning tree algorithms are useful for graph based parsing methods.

# Chapter 3

# TRANSITION-BASED DEPENDENCY PARSING

In transition-based dependency parsing, the model generates a sequence of transitions to find the dependency graph with respect to the given sentence. The learned model should be able to predict the transition step to be applied from the current state, so that a sequence of transitions can be found which leads to the optimal dependency graph for the given sentence. The method describes a model using specifications to construct dependency trees and benefits from given data and previous solutions for guessing the following transition. [Kırnap, 2018] applies dependency parsing methods to construct a graph which represents the grammatical structure for each sentence. The graph includes arcs connecting pairs of words to show that there is a grammatical relation between them. Two major findings presented in the paper are context embeddings and tree-stack LSTM. The paper claims that the ratio of correct predictions increases when context embeddings are used to represent words of each sentence. In addition, instead of defining the features manually, methods for automatizing selection of features are introduced.

[Kırnap et al., 2017] introduces creators of parsing models to be compelled to extend the set of attributes by manually including compositions. This is because unions of properties which can benefit choices for parsing are not able to be illustrated with shallow linear models. The creator of a deep model only needs to implement basic attributes since unions of features which help solving the problem can be illustrated and learned by deep models. [Kırnap et al., 2017] trains a bidirectional LSTM model using texts with tokens. The model is used for designing attributes for the parser to estimate words in a sentence. [Kırnap, 2018] also declares that transition based parsing systems usually find the resulting graph faster compared to graph-based systems.

[Kiperwasser and Goldberg, 2016] state that the means of specifying the structure and existing transitions for each system cause the distinction among those transition systems. Such a transition-based parsing system is proposed by [Kiperwasser and Goldberg, 2016] and consists of a group of patterns and transitions over these patterns. The patterns applied by the parser depend on the sentence to be processed. The parsing model finds a dependency graph for each sentence by applying several transition steps based on given patterns. According to the properties of the patterns used, the parser decides which transitions to apply in each step. The model achieves this by using appropriate patterns to prepare each sentence for parsing. Each pattern represents each configuration as a vector. The parsing model can process the sentences when they are represented using vectors.

Transition-based dependency parsing benefits form a stack used for storing the words being processed, a buffer for keeping current position of the parser on the sentence and an arc set for storing the dependency tree being constructed. Transition-based parsing algorithms include steps operating on these structures to find the most accurate dependency graph for given sentence. The method parsing algorithms modify the structures while searching for the dependency graph depends on the type of algorithm. Two of the most common types of algorithms implemented for transition-based parsing are arc- standard and arc-eager parsing algorithms.

## *3.1 Arc-Standard Parsing Algorithm*

There are three kinds of transitions in arc-standard parsing algorithm which operate on defined structures:

The first operation is left-arc transition, which connects a head word with its dependent on the left on the sentence. In arc-standard parsing algorithm, left-arc transition links the first word in the buffer with the topmost word in the stack, if the connection between them is appropriate. The topmost word is also removed from the stack since its head word has been found. The dependency relation also identifies the first word in the buffer, as head of the topmost word in the stack is the dependency

relation added to the arc set. For left-arc transition to be an available option, there must be at least one word in both the stack and the buffer, and the topmost word in the stack must not be the root of the sentence.

In contrast to the left-arc algorithm, right-arc transition connects a head word with its dependent on the right on the sentence. The principle is similar with left-arc transition, as both make a link between the topmost word in the stack with the first word in the buffer, but the word in the stack is the head word in right-arc transition. At the end of the step, the first word in the buffer is replaced with the topmost word in the stack. As it is the case with left-arc transition, both the stack and the buffer must contain at least one word for the right-arc transition to be an available option.

The final option is shift transition, which moves the first word in the buffer to the top of the stack.

## *3.2 Arc-Eager Parsing Algorithm*

Arc-eager parsing algorithm consists of four operations defined on the structures:

Left-arc transition links the topmost word in the stack as the dependent of the first word in the buffer, then removes the topmost word from the stack because its head word is found. On the other hand, right-arc transition, connects the topmost word in the stack as the head of the first word in the buffer. The first word in the buffer is then moved to the stack, on top of its head word. Shift transition moves the first word in the buffer to the top of the stack, without connecting any words. Finally, if the topmost word already has a dependent word in the sentence, reduce transition removes that word from the stack.

## *3.3 Our Approach*

In our submission [Kırnap et al., 2017] for CoNLL 2017 Shared Task [Zeman et al., 2017], we used the hybrid model proposed by [Kuhlmann et al., 2011]. The hybrid model includes transitions from both arc-standard and arc-eager parsing

algorithms. The hybrid model can therefore be considered as a mixture of said algorithms.

The hybrid model consists of three types of transitions. The first operation is shift transition. It moves the first word in the buffer to the top of the stack, in the same way with arc-standard and arc-eager parsing algorithms. Another operation is left-arc transition, which attaches the first word in the buffer to the topmost word in the stack as its head word, and removes the topmost word from the stack. Finally, right-arc transition attaches the second topmost word in the stack to the topmost word as its head.

Chapter 4

# GRAPH-BASED DEPENDENCY PARSING

Researchers want to figure out whether dependency parsing models can benefit from already existing graph algorithms. This interest has led to development of models using graph algorithms for dependency parsing.

For each sentence, graph-based dependency parsing models evaluate arcs representing relations between words in the sentence. In graph-based dependency parsing, the model tries to directly generate the most convenient dependency graph from the given sentence. The model forms an evaluation metric to determine how suitable a dependency graph is for each sentence. Thus, the questions are how to define the best criterion for evaluating the dependency graphs for a given sentence, and how to find the most appropriate dependency graph for each sentence according to the decided criterion.

Graph-based dependency parsers centralize on scoring each tree they process. Models usually score the tree for a sentence according to the probability of the tree representing accurate dependency analysis of that sentence.

[McDonald and Pereira, 2006] expresses ways of creating a dependency parser, the nature of the structure of the parsing model, and how to process new sentences using the produced model. The training data, which is formed by texts having parsed sentences is said to benefit the learning algorithm described during the process.

In graph-based parsing, each possible relation between two words of a sentence is represented by an arc, and each arc is assigned a score. The score of each arc is determined by its probability of being present in the dependency tree resulting from the correct analysis of the sentence. Graph-based parsing algorithms try to find the highest scoring dependency tree for each sentence, in which every word, except the

root has a parent. Also, no cycles should exist in a dependency tree and there should always be a connected undirected path between every two words of a sentence. Graph-based parsing algorithms must be able to achieve these, and they can be categorized as projective or non-projective algorithms depending on the type of dependency graphs they can generate for a sentence.

## *4.1 Chu-Liu-Edmonds Algorithm*

Graph-based parsing algorithms can generate both projective and non-projective dependency trees after analyzing a sentence. Chu-Liu-Edmonds algorithm [Chu, 1965], [Edmonds, 1967] is a non-projective parsing algorithm. The algorithm first chooses the arc with the highest score directed to each word. If there is a cycle in generated structure, the algorithm considers the words contributing to the cycle as a single word. It then tries to construct a tree whose nodes are the words included in the cycle. After solving cycle problem, the algorithm recalculates the highest score considering the words contributing to the cycle as a single word. For each word, the number of steps the algorithm takes for finding the most probable head word is proportional to the square of the number of words in the sentence. The number of cycles in a dependency graph of a sentence also cannot be higher than the number of words, since each word can be part of a single cycle. The number of steps the algorithm can take is therefore bounded by the cube of number of words in the sentence.

## *4.2 Experiments & Results*

Stanford's graph based dependency parsing model [Dozat et al., 2017] is used as baseline while running experiments. Different embeddings are implemented on top of baseline model to observe their effect on parsing performance. Experiments are run on English Web Treebank in Universal Dependencies version 2.2 [Nivre et al., 2018] dataset using AllenNLP deep learning framework [Gardner et al., 2017]. During each experiment, 100 dimensional part-of-speech tagger is used, data is read and processed in batches of 128 sentences, and model is trained for 50 epochs.

| GloVe | Without Finetuning | With Finetuning |
|---|---|---|
| **Training UAS**[1] | 97.94 | 98.13 |
| **Training LAS**[2] | 97.08 | 97.43 |
| **Validation UAS** | 90.21 | 90.54 |
| **Validation LAS** | 88.02 | 88.35 |
| **Best epoch** | 41 | 41 |
| **Best Validation UAS** | 90.23 | 90.65 |
| **Best Validation LAS** | 88.06 | 88.39 |

Table 4.1: Results using GloVe embeddings with Stanford's graph based dependency parsing model.

[Dozat et al., 2017] uses GloVe embeddings [Pennington et al., 2014] with finetuning in their model. We also tested the model with GloVe embeddings initially. During experiments, we used 100 dimensional GloVe embeddings containing 6 billion tokens with a vocabulary size of 400.000. GloVe embeddings used in the experiments are pretrained on English Gigaword Fifth Edition dataset [Parker et al., 2011] and Wikipedia dumps. The results of our experiments using GloVe embeddings are shown in Table 4.1.

SCODE word embeddings [Maron et al., 2010] are also used with Stanford's graph based model, instead of GloVe embeddings. The embeddings are pretrained on Reuters Corpus Volume I (RCV1) dataset containing 190 million words [Lewis et al., 2004] and Wikipedia dumps, as instructed in [Cirik, 2014] and [Cirik and Yuret, 2014], and observed results are compared and shown in Tables 4.2, 4.3, 4.4 and 4.5.

Another experiment is performed using word2vec embeddings [Mikolov et al., 2013a], which are pretrained on part of Google News dataset containing about 100 billion words. The model contains 300 dimensional vectors for 3 million words and phrases obtained using a simple data-driven approach described in [Mikolov et al., 2013b]. Results obtained using word2vec embeddings

| **Text field embedder dimension** | 25 | 50 | 100 | 200 |
|---|---|---|---|---|
| **Training UAS** | 97.38 | 97.70 | 98.03 | 98.25 |
| **Training LAS** | 96.36 | 96.81 | 97.24 | 97.47 |
| **Validation UAS** | 89.86 | 90.20 | 90.68 | 90.58 |
| **Validation LAS** | 87.40 | 87.94 | 88.37 | 88.34 |

Table 4.2: Results using SCODE embeddings with various dimensions, each pretrained on RCV1 dataest and without finetuning

| **Text field embedder dimension** | 25 | 50 | 100 | 200 |
|---|---|---|---|---|
| **Training UAS** | 97.93 | 98.32 | 98.56 | 98.86 |
| **Training LAS** | 97.14 | 97.60 | 97.95 | 98.36 |
| **Validation UAS** | 91.13 | 91.36 | 91.19 | 90.98 |
| **Validation LAS** | 88.86 | 89.24 | 89.02 | 88.76 |

Table 4.3: Results using SCODE embeddings with various dimensions, each pretrained on RCV1 dataest and with finetuning

| **Text field embedder dimension** | 25 | 50 | 100 | 200 |
|---|---|---|---|---|
| **Training UAS** | 97.48 | 97.80 | 98.10 | 98.27 |
| **Training LAS** | 96.41 | 96.95 | 97.30 | 97.54 |
| **Validation UAS** | 89.90 | 90.15 | 90.56 | 90.63 |
| **Validation LAS** | 87.56 | 87.88 | 88.33 | 88.38 |

Table 4.4: Results using SCODE embeddings with various dimensions, each pretrained on Wikipedia dumps and without finetuning

| **Text field embedder dimension** | 25 | 50 | 100 | 200 |
|---|---|---|---|---|
| **Training UAS** | 98.10 | 98.28 | 98.61 | 98.91 |
| **Training LAS** | 97.32 | 97.55 | 98.00 | 98.37 |
| **Validation UAS** | 90.86 | 91.22 | 91.17 | 91.11 |
| **Validation LAS** | 88.57 | 88.97 | 89.01 | 88.81 |

Table 4.5: Results using SCODE embeddings with various dimensions, each pretrained on Wikipedia dumps and with finetuning

| **Word2vec** | **Without Finetuning** | **With Finetuning** |
|---|---|---|
| **Training UAS** | 98.41 | 98.80 |
| **Training LAS** | 97.79 | 98.29 |
| **Validation UAS** | 91.31 | 91.59 |
| **Validation LAS** | 89.20 | 89.45 |

Table 4.6: Results using Word2vec embeddings with Stanford's graph based dependency parsing model.

with Stanford's graph based parsing model are shown in Table 4.6.

Fasttext word embeddings [Mikolov et al., 2018] are also tested with the baseline model. During experiments, 300 dimensional Fasttext word embeddings are used. According to [Mikolov et al., 2018], 1 million Fasttext word vectors having 16 billion tokens are trained on Wikipedia 2017, UMBC webbase corpus [Han et al., 2013] and statmt.org news dataset. Table 4.7 shows results obtained using given Fasttext embeddings with Stanford's model.

Several word embeddings from [Mikolov et al., 2018] are tested with Stanford's parsing model during experiments. 1 million Fasttext word vectors having 16 billion tokens trained with subword infomation on Wikipedia 2017, UMBC webbase corpus [Han et al., 2013] and statmt.org news dataset are also analysed using Stanford's graph based parsing model as a part of the experiments. The results are shown in Table 4.8.

| **Fasttext 1M** | **Without Finetuning** | **With Finetuning** |
|---|---|---|
| **Training UAS** | 98.32 | 98.86 |
| **Training LAS** | 97.59 | 98.37 |
| **Validation UAS** | 91.18 | 91.01 |
| **Validation LAS** | 89.11 | 88.95 |
| **Best epoch** | 47 | 48 |
| **Best Validation UAS** | 91.28 | 91.39 |
| **Best Validation LAS** | 89.18 | 89.26 |

Table 4.7: Results using 1 million Fasttext word embeddings having 16B tokens, trained on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset.

| **Fasttext 1M subword** | **Without Finetuning** | **With Finetuning** |
|---|---|---|
| **Training UAS** | 98.06 | 99.02 |
| **Training LAS** | 97.24 | 98.52 |
| **Validation UAS** | 90.33 | 90.12 |
| **Validation LAS** | 88.02 | 87.62 |
| **Best epoch** | 42 | 47 |
| **Best Validation UAS** | 90.45 | 90.09 |
| **Best Validation LAS** | 88.30 | 87.76 |

Table 4.8: Results using 1 million Fasttext word vectors having 16B tokens, trained with subword infomation on Wikipedia 2017, UMBC webbase corpus and statmt.org news dataset.

| **Fasttext Crawl 2M** | **Without Finetuning** | **With Finetuning** |
|---|---|---|
| **Training UAS** | 98.45 | 98.78 |
| **Training LAS** | 97.80 | 98.25 |
| **Validation UAS** | 91.58 | 91.62 |
| **Validation LAS** | 89.40 | 89.57 |
| **Best epoch** | 42 | 50 |
| **Best Validation UAS** | 91.62 | 91.62 |
| **Best Validation LAS** | 89.41 | 89.57 |

Table 4.9: Results using 2 million Fasttext word vectors having 600B tokens, trained on Common Crawl.

Another experiment related to Fasttext word embeddings is performed using 2 million word vectors with 600 billion tokens trained on Common Crawl on top of Stanford's graph based parser. Table 4.9 shows results of said experiment.

2 million Fasttext word vectors having 600 billion tokens trained with subword information on Common Crawl are also tested with the baseline model during experiments, of which the results are shown in Table 4.10.

We also used ELMo embeddings [Peters et al., 2018] with Stanford's graph based dependency parsing model as a part of our experiments. ELMo embeddings used for experiments are pre-trained on 1 Billion Word Benchmark [Chelba et al., 2013], having about 800 million tokens of news crawl data from WMT 2011. Model details are given in Table 4.11, and results obtained with ELMo embeddings are shown in Table 4.12.

### *4.3 Discussion & Conclusion*

Based on obtained results, Fasttext word embeddings yield highest labeled attachment score on validation data, while word2vec embeddings being close second. Both Fasttext and word2vec are static embeddings, which map a word to the same word vector

| **Fasttext Crawl 2M subword** | **Without Finetuning** | **With Finetuning** |
|---|---|---|
| **Training UAS** | 98.35 | 99.01 |
| **Training LAS** | 97.66 | 98.51 |
| **Validation UAS** | 91.04 | 90.92 |
| **Validation LAS** | 88.88 | 88.69 |
| **Best epoch** | 47 | 50 |
| **Best Validation UAS** | 91.26 | 90.92 |
| **Best Validation LAS** | 89.09 | 88.69 |

Table 4.10: Results using 2 million Fasttext word vectors having 600B tokens, trained with subword information on Common Crawl.

| **Hyperparameter** | **Value** |
|---|---|
| Input size | 1224 |
| Hidden size | 300 |
| Number of layers | 8 |
| Batch size | 128 |
| Input dropout | 0.3 |
| Dropout | 0.5 |
| Reccurent dropout probability | 0.1 |
| Number of epochs | 50 |

Table 4.11: Hyperparameters used while testing the model with ELMo embeddings.

| **ELMo** | **Without Finetuning** | **With Finetuning** |
|---|---|---|
| **Training UAS** | 98.58 | 98.62 |
| **Training LAS** | 97.94 | 97.98 |
| **Validation UAS** | 90.57 | 91.14 |
| **Validation LAS** | 88.40 | 88.69 |
| **Best epoch** | 44 | 47 |
| **Best Validation UAS** | 90.92 | 91.27 |
| **Best Validation LAS** | 88.66 | 88.87 |

Table 4.12: Results using ELMo embeddings with Stanford's graph based parsing model.

in all sentences. As 300 dimensional word embeddings are used for both embeddings during experiments, it can be said that embedding size of 300 yields highest labeled attachment scores compared to other sizes tested during experiments. However, this outcome is relative to several factors such as model architecture, hyperparameters and the dataset used and experiments with different settings can lead to different results.

On the other hand, ELMo embeddings are dynamic unlike other tested embeddings, which means word vectors generated by ELMo embeddings do not only depend on the word itself, but also on the sentence being processed. And, as a language model pretrained on a large corpora, ELMo based models can have better results with higher number of parameters and larger training data than which are used during experiments.

Notes to Chapter 4

**1** Unlabeled Attachment Score

**2** Labeled Attachment Score

# Chapter 5

# STRUCTURED BIAFFINE PARSER

As Koç University Graph Based Parsing team, we have developed a graph-based dependency parsing system for the CoNLL 2018 Shared Task [Zeman et al., 2018]. We called our Graph-Based Parsing model SParse [Önder et al., 2018], and our model extends the state-of-the-art biaffine parser [Dozat and Manning, 2016] with a structural meta-learning module, that combines local and global label predictions.

The parsing model has been trained and run on Universal Dependencies datasets [Nivre et al., 2018] and has 87.48% LAS, 78.63% MLAS, 78.69% BLEX and 81.76% CLAS [Nivre and Fang, 2017] score on the Italian-ISDT dataset and has 72.78% LAS, 59.10% MLAS, 61.38% BLEX and 61.72% CLAS score on the Japanese-GSD dataset in our official submission. Other corpora are also evaluated, for whom we present our results.

## *5.1 Introduction*

End-to-end learning with neural networks has proven to be effective in parsing natural language [Kiperwasser and Goldberg, 2016]. Graph-based dependency parsers [McDonald et al., 2005] represent dependency scores between words as a matrix representing a weighted fully connected graph, from which a spanning tree algorithm extracts the best parse tree. This setting is very compatible with neural network models that are good at producing matrices of continuous numbers.

Compared to transition based parsing [Kiperwasser and Goldberg, 2016], which was the basis of our previous entry [Kırnap et al., 2017], graph-based parsers have the disadvantage of producing $n^2$ entries for parsing an $n$-word sentence. Furthermore, algorithms used to parse these entries can be even more complex than $O(n^2)$.

However, graph-based parsers allow easy-to-parallelize static architectures rather than sequential decision mechanisms and are able to parse non-projective sentences. Non-projective graph-based parsing is the core of the winning entry [Dozat et al., 2017] in CoNLL 2017 Shared Task [Zeman et al., 2017].

Neural graph-based parsers can be divided into two components: encoder and decoder. The encoder is responsible for representing the sentence as a sequence of continuous feature vectors. The decoder receives this sequence and produces the parse tree, by first creating a graph representation and then extracting the maximum spanning tree (MST).

A bidirectional RNN (bi-RNN) is used to produce a contextual vector for each word in a sentence. Use of bi-RNNs is the defacto standard in dependency parsing, as it allows representing each word conditioned on the whole sentence. The main contribution in the encoder part is to the word embeddings feeding the bi-RNN. The model uses word vectors coming from a language model pretrained on very large language corpora, similar to [Kırnap et al., 2017]. Word embeddings are extended with learnable embeddings for UPOS tags, XPOS tags and FEATs where applicable.

The decoder can be viewed as a more structured version of the state-of-the-art biaffine decoder of [Dozat et al., 2017], where the goal is to condition the label-seeking units to a parse-tree instead of simple local predictions. A meta-learning module is proposed that allows structured and unstructured predictions to be combined as a weighted sum. This additional computational complexity is paid off by the simple word-level model in the encoder part. It is called the structured meta-biaffine decoder or shortly SMeta. The model is implemented using Knet deep learning framework [Yuret, 2016] in Julia language [Bezanson et al., 2017].

## *5.2 Related work*

[Kiperwasser and Goldberg, 2016] use trainable BiLSTMs to represent features of each word, instead of defining the features manually. They formulated the structured prediction using hinge loss based on the gold parse tree and parsed scores.

[Dozat and Manning, 2016] propose deep biaffine attention combined with the parsing model of [Kiperwasser and Goldberg, 2016], which simplifies the architecture by allowing implementation with a single layer instead of two linear layers.

Stanford's Graph-based Neural Dependency Parser [Dozat et al., 2017] at the CoNLL 2017 Shared Task [Zeman et al., 2017] is implemented with four ReLU layers, two layers for finding heads and dependents of each word, and two layers for finding the dependency relations for each head-dependent pair. The outputs are then fed into two biaffine layers, one for determining the head of the word, and another for determining the dependency relation of head-dependent pair.

We, as SParse team, propose a dependency parsing model based on the graph-based parser by [Dozat and Manning, 2016]. We are adding a meta-biaffine decoder layer, similar to the tagging model proposed by [Bohnet et al., 2018], for computing the arc labels based on the full tree constructed from the unlabeled arc scores instead of computing them independently.

Our parsing model uses pretrained word embeddings from [Kırnap et al., 2017]. Our parser uses the same language model with [Kırnap et al., 2017], in which graph based-parsing algorithms are applied. However, a transition-based parsing model is given in [Kırnap et al., 2017]. Therefore, some adaptations are made on the features proposed by [Kırnap et al., 2017] in order to use them in a graph based parsing model. We did not use contextual features coming from the language model or features related to words in stack and buffer. Instead, we trained a three-layer BiLSTM from scratch to encode contextual features.

## *5.3 Model*

In this section, we depict important aspects of our architecture which is shown in Figure 5.1. We discuss encoder and decoder separately and then give the model hyper-parameters used.

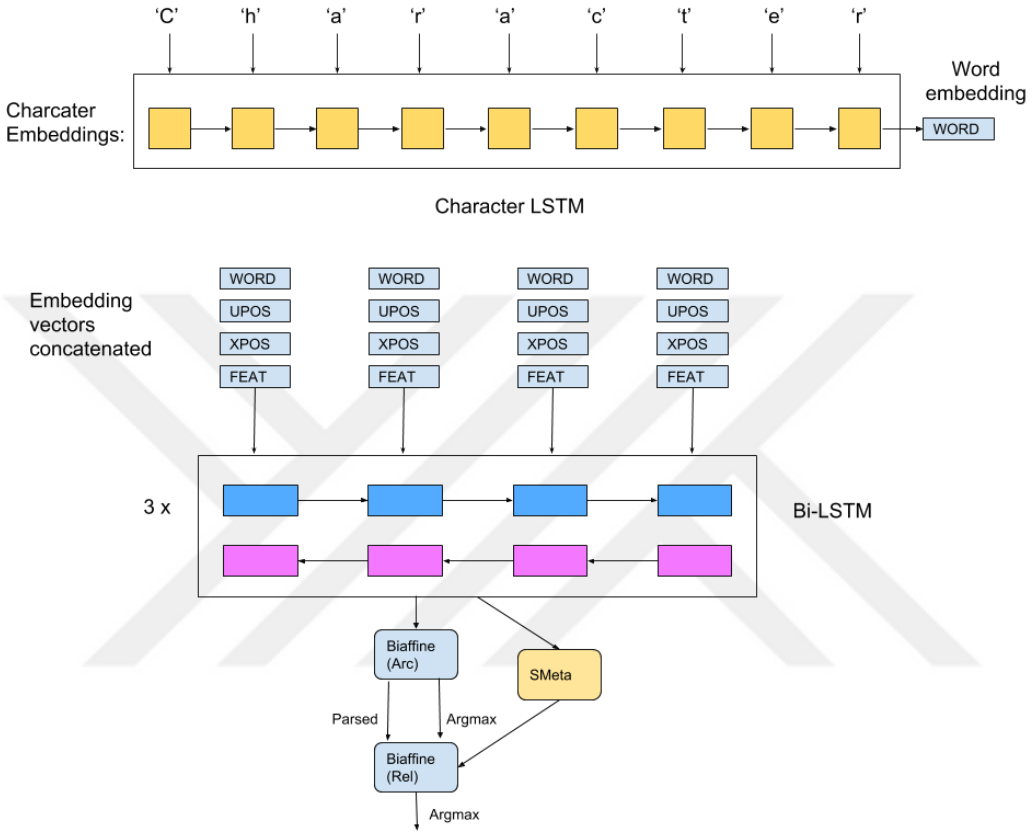


Figure 5.1: Overall model architecture.

### *5.3.1 Encoder*

*Word Model*

We used four main features to represent each word in a sentence: a pre-trained word embedding, UPOS tag embedding, XPOS tag embedding and FEAT embedding.

Pre-trained words come from the language model in [Kırnap et al., 2017]. This model represents each word using a character-level LSTM, which is a suitable setting for morphologically rich languages, as shown in [Dozat et al., 2017]. We use the word vectors without further training.

UPOS and XPOS tag embeddings are represented by vectors randomly initialized using unit Gaussian distribution.

Morphological features, also called FEATs, are different in the sense that there are zero or more FEATs for each word. We follow a simple strategy: we represent each FEAT using a randomly initialized vector and add all FEAT embeddings for each word. We simply used zero for word vectors without any morphological features.

For practical reasons, we also needed to represent ROOT word of a sentence. We do so by randomly initializing a word embedding and setting all other embeddings to zero.

At test time, we used tags and morphological features produced by MorphNet [Dayanık et al., 2018]. For languages where this model is not available, we directly used UDPipe results [Straka et al., 2016].

*Sentence Model*

We used a three-layer bidirectional LSTM to represent a sentence. We used the hidden size of 200 for both forward and backward LSTMs. Dropout [Srivastava et al., 2014] is performed at the input of each LSTM layer, including the first layer. Our LSTM simply use $n$th hidden state for $n$th word, different from the language model in [Kırnap et al., 2017].

The language model discussed in the previous section also provides context embeddings. We performed experiments for combining our own contextual representation with this representation using various concatenation and addition strategies, but we observed poorer performance in terms of generalization. Also, using the language model directly as a feature extractor led to unsatisfactory performance, different from the transition-based entry of our institution [Kırnap et al., 2017].

### *5.3.2 Decoder*

*Structured Meta-Biaffine Decoder (SMeta)*

Deep biaffine decoder [Dozat and Manning, 2016] is the core of the winning entry [Dozat et al., 2017] of CoNLL 2017 Shared task [Zeman et al., 2017], so we used this module as our starting point. Biaffine architecture is computationally efficient and

can be used with easy-to-train softmax objective, different from harder-to-optimize hinge loss objectives as in [Kiperwasser and Goldberg, 2016].

Similar to [Dozat et al., 2017], we produce four different hidden vectors, two for arcs and two for relations (or labels). Formally

$$\mathbf{h}_i^{(arc-dep)} = \mathbf{MLP}^{(arc-dep)}(\mathbf{h}_i)$$

$$\mathbf{h}_i^{(arc-head)} = \mathbf{MLP}^{(arc-head)}(\mathbf{h}_i)$$

$$\mathbf{h}_i^{(rel-dep)} = \mathbf{MLP}^{(rel-dep)}(\mathbf{h}_i)$$

$$\mathbf{h}_i^{(rel-head)} = \mathbf{MLP}^{(rel-head)}(\mathbf{h}_i)$$

where $\mathbf{h}_i$ represents $i$th hidden state of the bi-LSTM embedding. The vectors correspond to arcs seeking their dependents, arcs seeking their heads, and corresponding relations. **MLP** can be any neural network module. Here, we simply use dense layers followed by ReLU activations, as in [Dozat et al., 2017].

Now, we perform the biaffine transformation to compute the score matrix representing the graph,

$$\mathbf{s}_i^{(arc)} = H^{(arc-head)} W^{(arc)} \mathbf{h}_i + H^{(arc-head)} \mathbf{b}^{\mathbf{T}(arc)}$$

where $H^{(arc-head)}$ represents matrix of $\mathbf{h}_i^{(arc-head)}$ vectors, $W^{(arc)}$ and $\mathbf{b}^{(arc)}$ are learnable weights.

Up to this point, our decoder is identical to the one in [Dozat et al., 2017]. The difference is in the computation of predicted arcs. We compute two different predictions:

$$y_i^{'l(arc)} = \arg\max_j s_{ij}^{(arc)}$$

$$\mathbf{p} = parse(S^{(arc)})$$

$$y_i^{'s(arc)} = p_i$$

Here $S^{(arc)}$ is the matrix of arc scores and $parse$ is a spanning tree algorithm that computes the indices of the predicted arcs. Now, we compute label scores using these

two predictions. First, we compute coefficient vector $\mathbf{k}$ using the bi-RNN encodings,

$$\mathbf{h}' = \frac{\sum_{i=1}^{n} \mathbf{h}_i}{n}$$

$$\mathbf{k} = W^{(meta)}\mathbf{h}' + \mathbf{b}^{(meta)}$$

where $n$ is the number of words in the sentence, $W$ and $\mathbf{b}$ are learned parameters. Averaging over time is inspired by the global average pooling operator in the vision literature [Lin et al., 2013], transforming temporal representation to a global one.

We now compute the weighted sum of label predictions using coefficient vector $\mathbf{k}$.

$$\mathbf{s}_i^{l(rel)} = \mathbf{h}_{y_i'^{l(arc)}}^{\mathrm{T}(rel-head)}\mathbf{U}^{(rel)}\mathbf{h}_i^{(rel-dep)} + W^{(rel)}cat(\mathbf{h}_i^{(rel-dep)}, \mathbf{h}_{y_i'^{l(arc)}}^{(rel-head)}) + \mathbf{b}^{(rel)}$$

$$\mathbf{s}_i^{s(rel)} = \mathbf{h}_{y_i'^{s(arc)}}^{\mathrm{T}(rel-head)}\mathbf{U}^{(rel)}\mathbf{h}_i^{(rel-dep)} + W^{(rel)}cat(\mathbf{h}_i^{(rel-dep)}, \mathbf{h}_{y_i'^{s(arc)}}^{(rel-head)}) + \mathbf{b}^{(rel)}$$

$$\mathbf{s}_i^{(rel)} = k_1\mathbf{s}_i^{l(rel)} + k_2\mathbf{s}_i^{s(rel)}$$

$$y_i'^{(rel)} = \arg\max_j s_{ij}^{(rel)}$$

where $\mathbf{U}^{(rel)}$, $W^{(rel)}$ and $\mathbf{b}^{(rel)}$ are learned parameters.

Our model is trained using sum of softmax losses similar to [Dozat et al., 2017].

*Parsing algorithms*

In our parsing model, Chu-Liu-Edmonds algorithm [Chu, 1965, Edmonds, 1967] and [Eisner, 1996]'s algorithm are used interchangeably, during both the training of parser models and parsing phase of test datasets. On the languages whose training dataset consists of more than 250,000 words, Chu-Liu-Edmonds algorithm is used for parsing since it has a complexity of $O(n^2)$, where $n$ is the number of words.

This approach allows us to train our models on relatively larger datasets in less amount of time, compared to the Eisner's algorithm whose time complexity is $O(n^3)$.

On training datasets having at most 250,000 words, Eisner's algorithm is used during both training and parsing phase. Eisners algorithm produces only projective trees and Chu-Liu-Edmonds algorithm produces both projective and non-projective trees. This means the number of possible trees Eisner's algorithm can generate is

fewer compared to Chu-Liu-Edmonds algorithm, so even though Eisner's algorithm has higher time complexity than Chu-Liu-Edmonds algorithm, parsing models are trained faster when Eisner's algorithm is used.

### *5.3.3 Hyperparameters*

We used a 150-dimensional tag and feature embeddings and 350-dimensional word embeddings for the word model. Bi-RNN sentence model has the hidden size of 200 for both forward and backward RNNs, producing 400-dimensional feature context vectors. We used the hidden size of 400 for arc MLPs and 100 for relation MLPs.

## ***5.4 Training***

We used Adam optimizer [Kingma and Ba, 2014] with its standard parameters. Based on dataset size, we trained the model for 25 to 100 epochs and selected the model based on its validation labeled attachment accuracy.

We sampled sentences with identical number of words in a minibatch. In training corpora that are sufficiently large, we sampled minibatches so that approximately 500 tokens exist in a single minibatch. We reduced this size to 250 for relatively small corpora. For very small corpora, we simply sample a constant number of sentences as a minibatch.

We saved best models with corresponding configurations, scores and optimization states during training for recovery. We then re-create the model files for the best models of each corpus by removing the optimization states.

MorphNet is the morphological analysis and disambiguation tool proposed by [Dayanık et al., 2018], which we used while training our parsing model. During training of the parser, CoNLL-U formatted training dataset files, which are produced by UDPipe [Straka et al., 2016], are given to MorphNet as input. Then, MorphNet applies its own morphological analysis and disambiguation, and new CoNLL-U formatted files produced by MorphNet are used by our parser.

Even though we used lemmatized and tagged outputs generated by MorphNet

while training our parser, we run our parser on outputs generated by UDPipe, due to time constrains during parsing.

### *5.5 Results and Discussion*

Our parser obtained results on 64 treebank test sets out of 82, which are shown in Table 5.1 and 5.2. Tests are done in TIRA [Potthast et al., 2014] machine allocated for the task. According to the results, we had an average LAS score of 57% on the 64 test sets on which our model is run. The MLAS score of our model is 46.40% and the BLEX score of our model is 49.17%.

According to the results, our model performs better at datasets with comparably larger training data. For instance, our model has around 90% LAS score on Catalan, Indian, Italian, Polish and Russian languages which have higher number of tokens in training data. Furthermore, our model performs relatively well in some low-resource languages like Turkish and Hungarian. However, on the datasets with very small or no training data, such as Japanese Modern, Russian Taiga and Irish IDT, we get lower scores. Hence, our model benefits from large amount of data during training process, but prediction with low resources remains as an issue for our model.

### *5.6 Contributions*

In this work, we proposed a new decoding mechanism, called SMeta, for graph-based neural dependency parsing. This architecture attempts to combine structured and unstructured prediction methods using meta-learning. We coupled this architecture with custom training methods and algorithms to evaluate its performance.

| **Dataset** | **LAS** | **MLAS** | **BLEX** | **Dataset** | **LAS** | **MLAS** | **BLEX** |
|---|---|---|---|---|---|---|---|
| ar_padt | 67.57% | 56.22% | 58.84% | hu_szeged | 73.84% | 56.03% | 63.31% |
| bg_btb | 85.85% | 76.28% | 74.73% | id_gsd | 76.88% | 65.34% | 64.89% |
| ca_ancora | 87.60% | 79.05% | 79.49% | it_isdt | 87.51% | 78.81% | 78.75% |
| cs_cac | 86.05% | 73.54% | 80.14% | it_postwita | 69.48% | 56.20% | 56.73% |
| cs_fictree | 84.21% | 71% | 76.86% | ja_modern | 22.91% | 8.47% | 9.73% |
| cs_pdt | 86.07% | 76.54% | 81.39% | ko_gsd | 75.52% | 69.15% | 63.18% |
| cs_pud | 81.80% | 68.19% | 75.10% | ko_kaist | 81.64% | 74.46% | 68.86% |
| cu_proiel | 67.80% | 56.51% | 60.93% | la_ittb | 77.66% | 68.09% | 73.91% |
| da_ddt | 77.90% | 68.14% | 68.64% | la_perseus | 47.45% | 29.99% | 32.47% |
| de_gsd | 70.65% | 34.50% | 60.40% | la_proiel | 64.29% | 51.80% | 58.05% |
| el_gdt | 83.65% | 66.76% | 70.14% | lv_lvtb | 71.52% | 57.26% | 60.20% |
| en_ewt | 77.87% | 68.57% | 70.83% | nl_alpino | 78.28% | 63.38% | 66.08% |
| en_gum | 75.51% | 63.95% | 63.52% | nl_lassysmall | 78.10% | 65.73% | 66.77% |
| en_lines | 73.97% | 65.15% | 66.06% | pl_lfg | 88.21% | 75.40% | 79.25% |
| en_pud | 81.31% | 69.85% | 73.18% | pl_sz | 83.01% | 65.10% | 73.34% |

Table 5.1: F1 scores achieved by our model.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| es_ancora | 86.94% | 79.14% | 79.58% | pt_bosque | 84.32% | 70.27% | 75.29% |
| et_edt | 77.45% | 69.58% | 66.05% | ro_rrt | 82.74% | 74.19% | 74.60% |
| eu_bdt | 73.84% | 60.49% | 66.38% | ru_syntagrus | 88.22% | 80.16% | 81.05% |
| fa_seraji | 81.74% | 75.15% | 71.93% | ru_taiga | 40.01% | 23.92% | 25.44% |
| fi_ftb | 76.62% | 66.28% | 62.88% | sk_snk | 77.81% | 56.11% | 62.23% |
| fi_tdt | 79.30% | 71.07% | 64.37% | sl_ssj | 78.46% | 64.93% | 70.50% |
| fr_gsd | 82.32% | 72.91% | 74.97% | sl_sst | 43.63% | 31.14% | 35.41% |
| fr_sequoia | 82.60% | 73.07% | 76.07% | sv_lines | 74.87% | 59.32% | 67.25% |
| fr_spoken | 64.62% | 53.15% | 54.18% | sv_pud | 71.66% | 43.65% | 55.60% |
| ga_idt | 35.84% | 13.07% | 16.77% | sv_talbanken | 79.06% | 70.55% | 71.16% |
| gl_ctg | 80.31% | 67.77% | 70.72% | tr_imst | 60.19% | 49.57% | 51.02% |
| got_proiel | 62.40% | 49.19% | 55.15% | ug_udt | 58.48% | 38.18% | 45.94% |
| grc_perseus | 61.82% | 34.17% | 41.16% | uk_iu | 75.93% | 57.81% | 64.84% |
| grc_proiel | 68.03% | 49.40% | 55.62% | ur_udtb | 78.41% | 51.28% | 64.94% |
| he_htb | 60.09% | 46.32% | 49.19% | vi_vtb | 41.27% | 34.73% | 36.95% |
| hi_hdtb | 87.66% | 70% | 80.47% | zh_gsd | 59.51% | 49.33% | 54.33% |
| hr_set | 80.86% | 60.65% | 72.36% | | | | |

Table 5.2: F1 scores achieved by our model (continued).

# Chapter 6

# CONCLUSION

Recent studies in dependency parsing involve mostly rule-based and data-driven approaches. Data-driven approaches have a notable increase in usage in last few years. This is due to two main reasons. First, the amount of accessible data used for natural language processing such as texts and articles increase, and such resources become easily accessible for most researchers. Second, deep leaning techniques are proven to be efficient in solving natural language processing problems. However, methods introduced in this work are not the only way of applying dependency parsing, and new techniques continuously become available as the research progresses.

# BIBLIOGRAPHY

[Bezanson et al., 2017] Bezanson, J., Edelman, A., Karpinski, S., and Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98.

[Bohnet et al., 2018] Bohnet, B., McDonald, R., Simoes, G., Andor, D., Pitler, E., and Maynez, J. (2018). Morphosyntactic tagging with a meta-bilstm model over context sensitive token encodings. *arXiv preprint arXiv:1805.08237*.

[Chelba et al., 2013] Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., Koehn, P., and Robinson, T. (2013). One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*.

[Chu, 1965] Chu, Y.-J. (1965). On the shortest arborescence of a directed graph. *Scientia Sinica*, 14:1396–1400.

[Cirik, 2014] Cirik, V. (2014). *Analysis of SCODE World Embeddings Based on Substitute Distributions in Supervised Tasks*. PhD thesis, Koç University.

[Cirik and Yuret, 2014] Cirik, V. and Yuret, D. (2014). Substitute based scode word embeddings in supervised nlp tasks. *arXiv preprint arXiv:1407.6853*.

[Dayanık et al., 2018] Dayanık, E., Akyürek, E., and Yuret, D. (2018). Morphnet: A sequence-to-sequence model that combines morphological analysis and disambiguation. *arXiv preprint arXiv:1805.07946*.

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

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

[Edmonds, 1967] Edmonds, J. (1967). Optimum branchings. *Journal of Research of the national Bureau of Standards B*, 71(4):233–240.

[Eisner, 1996] Eisner, J. M. (1996). Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 340–345. Association for Computational Linguistics.

[Gardner et al., 2017] Gardner, M., Grus, J., Neumann, M., Tafjord, O., Dasigi, P., Liu, N. F., Peters, M., Schmitz, M., and Zettlemoyer, L. S. (2017). Allennlp: A deep semantic natural language processing platform.

[Han et al., 2013] Han, L., Kashyap, A. L., Finin, T., Mayfield, J., and Weese, J. (2013). Umbc_ebiquity-core: Semantic textual similarity systems. In *Second Joint Conference on Lexical and Computational Semantics (* SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 44–52.

[Kingma and Ba, 2014] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kiperwasser and Goldberg, 2016] Kiperwasser, E. and Goldberg, Y. (2016). Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

[Kırnap, 2018] Kırnap, Ö. (2018). *Transition Based Dependency Parsing with Deep Learning*. PhD thesis, Koç University.

[Kırnap et al., 2017] Kırnap, O., Önder, B. F., and Yuret, D. (2017). Parsing with context embeddings. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 80–87, Vancouver, Canada. Association for Computational Linguistics.

[Kübler et al., 2009] Kübler, S., McDonald, R., and Nivre, J. (2009). *Dependency parsing*. Synthesis lectures on human language technologies. Morgan & Claypool, US.

[Kudo and Matsumoto, 2002] Kudo, T. and Matsumoto, Y. (2002). Japanese dependency analysis using cascaded chunking. In *proceedings of the 6th conference on Natural language learning-Volume 20*, pages 1–7. Association for Computational Linguistics.

[Kuhlmann et al., 2011] Kuhlmann, M., Gómez-Rodríguez, C., and Satta, G. (2011). Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 673–682. Association for Computational Linguistics.

[Lewis et al., 2004] Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. (2004). Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research*, 5(Apr):361–397.

[Lin et al., 2013] Lin, M., Chen, Q., and Yan, S. (2013). Network in network. *arXiv preprint arXiv:1312.4400*.

[Maron et al., 2010] Maron, Y., Lamar, M., and Bienenstock, E. (2010). Sphere embedding: An application to part-of-speech induction. In *Advances in Neural Information Processing Systems*, pages 1567–1575.

[McDonald and Pereira, 2006] McDonald, R. and Pereira, F. (2006). *Discriminative learning and spanning tree algorithms for dependency parsing*. University of Pennsylvania.

[McDonald et al., 2005] McDonald, R., Pereira, F., Ribarov, K., and Hajič, J. (2005). Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., Dean, J., Sutskever, L., and Zweig, G. (2013a). word2vec. *URL https://code.google.com/p/word2vec*.

[Mikolov et al., 2018] Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., and Joulin, A. (2018). Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

[Mikolov et al., 2013b] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

[Nivre et al., 2018] Nivre, J. et al. (2018). Universal Dependencies 2.2. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University, Prague, http://hdl.handle.net/11234/1-2837.

[Nivre and Fang, 2017] Nivre, J. and Fang, C.-T. (2017). Universal dependency evaluation. In *Proceedings of the NoDaLiDa 2017 Workshop on Universal Dependencies (UDW 2017)*, pages 86–95.

[Önder et al., 2018] Önder, B., Gümeli, C., and Yuret, D. (2018). SParse: Koç university graph-based parsing system for the conll 2018 shared task. In *Proceedings*

*of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 216–222, Brussels, Belgium. Association for Computational Linguistics.

[Parker et al., 2011] Parker, R., Graff, D., Kong, J., Chen, K., and Maeda, K. (2011). English gigaword fifth edition ldc2011t07 (tech. rep.). Technical report, Technical Report. Linguistic Data Consortium, Philadelphia.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

[Peters et al., 2018] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proc. of NAACL*.

[Potthast et al., 2014] Potthast, M., Gollub, T., Rangel, F., Rosso, P., Stamatatos, E., and Stein, B. (2014). Improving the reproducibility of PAN's shared tasks: Plagiarism detection, author identification, and author profiling. In Kanoulas, E., Lupu, M., Clough, P., Sanderson, M., Hall, M., Hanbury, A., and Toms, E., editors, *Information Access Evaluation meets Multilinguality, Multimodality, and Visualization. 5th International Conference of the CLEF Initiative (CLEF 14)*, pages 268–299, Berlin Heidelberg New York. Springer.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

[Straka et al., 2016] Straka, M., Hajič, J., and Straková, J. (2016). UDPipe: trainable pipeline for processing CoNLL-U files performing tokenization, morphological analysis, POS tagging and parsing. In *Proceedings of the 10th International Con-*

*ference on Language Resources and Evaluation (LREC 2016)*, Portoro, Slovenia. European Language Resources Association.

[Yamada and Matsumoto, 2003] Yamada, H. and Matsumoto, Y. (2003). Statistical dependency analysis with support vector machines. In *Proceedings of the Eighth International Conference on Parsing Technologies*, pages 195–206.

[Yuret, 2016] Yuret, D. (2016). Knet: beginning deep learning with 100 lines of julia. In *Machine Learning Systems Workshop at NIPS 2016*.

[Zeman et al., 2018] Zeman, D., Hajič, J., Popel, M., Potthast, M., Straka, M., Ginter, F., Nivre, J., and Petrov, S. (2018). CoNLL 2018 shared task: Multilingual parsing from raw text to universal dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21, Brussels, Belgium. Association for Computational Linguistics.

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