NEURAL DEPENDENCY PARSING FOR TURKISH

TÜRKÇE İÇİN NÖRAL BAĞLILIK AYRIŞTIRMA

SALİH TUÇ

ASSİST. PROF. DR. BURCU CAN BUĞLALILAR
Supervisor

Submitted to
Graduate School of Science and Engineering of Hacettepe University
as a Partial Fulfillment to the Requirements
for the Award of the Degree of Master of Science
in Computer Engineering.

2020

ÖZET

TÜRKÇE İÇİN NÖRAL BAĞLILIK AYRIŞTIRMA

Salih Tuç

Yüksek Lisans, Bilgisayar Mühendisliği
Tez Danışmanı: Dr. Öğr. Burcu CAN BUĞLALILAR
Haziran 2020, 66 sayfa

Bağlılık ayrıştırma, sözcükler arasındaki sözdizimsel ve anlamsal ilişkilerin belirlenerek dilbilgisel yapıların ortaya çıkarılmasını içerir. Uzun dönemli bağlılıkların çıkarılması ve sözlük dışı sözcüklerin meydana getirdiği sorunlardan ötürü bağlılık ayrıştırmada henüz istenen başarı elde edilememiştir. Mevcut sorunlar Türkçe iiçin de geçerli olup, özellikle sondan eklemeli yapısı gereği sözlük dışı sözcük oranı diğer dillere göre nispeten daha fazladır.

Şimdiye kadar yapılan çalışmalar, Tekrarlı Sinir Ağlarının uzun dizilerde başarılı olamadığını göstermiştir. Bu tezde önerdiğimiz nöral model, kodlayıcı-kod çözücü yaklaşımını, Transformer ağı üzerine kurulu bir kodlayıcı ve Yığıt Tabanlı İşaretçi Ağı üzerine kurulu bir kod çözücüyle gerçekleştirmektedir. Sözlük dışı sözcük problemi için ise sözcüklerin karakter tabanlı österimleri kullanılmaktadır. hem Türkçe, hem de İngilizce için gerçekleştirilen deneyler, önerilen modelin diğer nöral modellere göre özellikle uzun cümlelerde daha başarılı olduğunu ve uzun dönemli bağlılıkları etkili bir şekilde bulabildiğini göstermektedir.

Anahtar Kelimeler: Bağlılık Ayrıştırma, Sözdizimi, Transformer, Yiğit Tabanlı İşaretçi Ağı

ABSTRACT

NEURAL DEPENDENCY PARSING FOR TURKISH

Salih Tuç

Master of Science, **Computer Engineering Department**
Supervisor: ASSIST. PROF. DR. BURCU CAN BUĞLALILAR
June 2020, 66 pages

Dependency Parsing is the task of finding the grammatical structure of a sentence by identifying syntactic and semantic relationships between words. The current accuracy of the dependency parsers is still not satisfying due to the long term dependencies and out-of-vocabulary (OOV) problem. Those problems also apply to Turkish because of the high percentage of OOV words due to its agglutinative morphological structure compared to other languages.

The recent work shows that Recurrent Neural Networks (RNNs) are not efficient for long sequences. The deep neural architecture that we propose in this thesis follows an encoder-decoder structure with an encoder based on a Transformer Network and a decoder based on a Stack Pointer Network. The character-level word embeddings are also integrated in the model to cope with the OOV problem. The results for both Turkish and English show that the proposed model performs better for long sentences and can identify long term dependencies more efficiently compared to other neural models.

Keywords: Dependency Parsing, Transformer, Pointer Network, Stack Pointer Network

CONTENTS

FIGURES

SYMBOLS AND ABBREVIATIONS

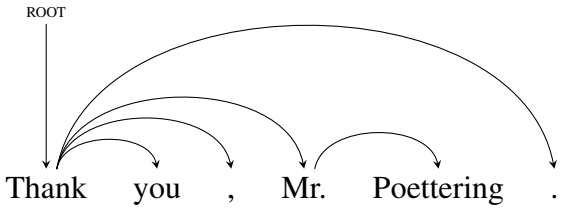
DP	Dependency Parsing
NLP	Natural Language Processing
POS	Part Of Speech
NER	Named Entity Recognition
CYK	Cocke–Younger–Kasami Algorithm
UD	Universal Dependencies
LSTM	Long short-term memory
Bi-LSTM	Bidirectional Long short-term memory
RNN	Recurrent Neural Networks
Seq2Seq	Sequence-to-sequence
PTB	Penn Treebank

1. INTRODUCTION

1.1. Overview

Dependency Parsing is the task of finding the grammatical structure of a sentence by identifying the syntactic and semantic relationships between words. Dependency parsing has been utilized in many other NLP tasks such as machine translation [10, 11], relation extraction [12, 13], named entity recognition [14, 15], information extraction [16, 17], all of which involve natural language understanding to an extent. Each dependency relation is identified between a head word and a dependent word that modifies the head word in a sentence. Although such relations are considered as syntactic, they are naturally built upon semantic relationships between words. For example, each dependent has a role of modifying its head word, which is a result of a completely semantic influence.

Dependency structures are represented either by hierarchical structures which are called **dependency trees**, or represented in the form of directed graphs, which are called **dependency graphs**. An example dependency graph for the sentence *Thank you, Mr. Pottering.* is given below:



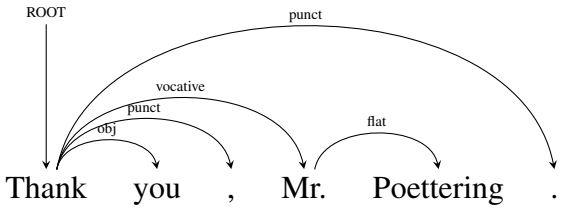
Syntactic relations between words are generally figured out with an arrow in a dependency tree, which connects each **head** word to a **dependent**. In other words, in a relation such as $I \rightarrow am$; "I" becomes the head and "am" becomes the dependent and the arrow between them states a **dependency** between the two words.

The *ROOT* token represents the root of the dependency tree (i.e. the starting point of dependency parsing or the head of the complete sentence). Even if the rules of dependency parsing will be discussed later, it is good to state here that every sentence must contain a ROOT token in its dependency tree.

It is also typical to define the type of each grammatical relation between a head and a dependent in a dependency structure. In the Universal Dependencies [18], there are 37 dependency relations defined. In the latest tag set in Universal Dependencies (UD v2.0), relations are split into four main categories (Core Arguments, Non-core dependents, Nominal dependents and Other) and nine sub-categories (Nominals, Clauses, Modifier Words, Function Words, Coordination, MWE, Loose Special and Other). Some of the prominent ones and their descriptions in UD are as follows:

- NSUBJ - Nominal subject *(Core arguments → Nominals)*
- OBJ - Object *(Core arguments → Nominals)*
- IOBJ - Indirect object *(Core arguments → Nominals)*
- CCOMP - Clausal complement *(Core arguments → Clauses)*
- XCOMP - Open clausal complement *(Core arguments → Clauses)*
- NMOD - Nominal modifier *(Nominal dependents → Nominals)*
- NUMMOD - Numeric modifier *(Nominal dependents → Nominals)*
- APPOS - Appositional modifier *(Nominal dependents → Nominals)*
- AMOD - Adjectival modifier *(Nominal dependents → Modifier Words)*
- DET - Determiner *(Nominal dependents → Function Words)*
- CASE - Prepositions and other case markers *(Nominal dependents → Function Words)*
- CONJ - Conjunct *(Other → Coordination)*
- CC - Coordinating conjunction *(Other → Coordination)*
- ROOT - Root *(Other → Other)*
- PUNCT - Punctuation *(Other → Other)*

The final dependency graph enriched with dependency relation types for the previous example, *"Thank you, Mr. Poettering."*, is given below:



Of course, not every sentence in a language is grammatically simple and short like this example. Discovering dependencies in longer sentences is one of the main challenges in dependency parsing. We will use the term *long term dependency* throughout the thesis to refer to orthographic distance and not structural distance. Hockenmaier and Steedman [19] define another term *long range dependency* in order to refer to the structural distance between dependencies. Our aim in this thesis is to find long term dependencies correctly, especially by considering orthographic distance.

Another challenge in dependency parsing is the out-of-vocabulary (OOV) words problem, where a test set generally contains some unseen words for which it gets harder to find dependencies.

1.2. Motivation

In this thesis, we mainly focus on Turkish grammar. Turkish has a free-order grammar and rich morphology, which makes dependency parsing even harder for Turkish language. Previous work on Turkish dependency parsing, does not perform well enough for especially long sentences.

Here, we propose a novel neural architecture based on Transformer Networks [8] and Stack Pointer Networks [9] to handle long-term dependencies problem. Recent Recurrent Neural Networks (RNNs) and sequence-to-sequence models based on such RNN structures have issues in finding relations between distant words in a sequence, since the data transferred from one end to another is prone to fade away. However, Transformer Networks do not have such sequential structures and all structure is built upon an attention mechanism, where relations between all word couples can be learned effectively. We utilize Transformer Networks to handle long-term dependencies in this thesis, which will be a novel language-independent neural dependency parser although our main focus is the Turkish language.

1.3. Research Questions

Our main aim is to find answers to the following research questions:

- Is it possible to overcome long-term dependencies problem in dependency parsing with a full attention mechanism, such as Transformer Networks without using any recurrent structure?
- Can a Transformer Network efficiently encode the grammatical structure of a sentence?
- Is it possible to mitigate the OOV problem by using character-level word embeddings in an agglutinative language like Turkish?

1.4. Thesis Outline

The thesis is structured as follows:

In **Chapter 2**, the essential background knowledge to ease the understanding of this thesis for any reader is given. The essential background knowledge is given under two sections: Linguistic background and deep learning background.

Literature review on dependency parsing for Turkish and other languages will be given in **Chapter 3**. In that section, models will be classified into graph-based and transition-based as two dependency parsing approaches in the literature. Moreover, the previous studies will be reviewed in terms of their methods, such as neural or probabilistic methods.

In **Chapter 4**, our proposed Transformer-Stack Pointer architecture will be explained in detail along with the mathematical details.

Finally, in **Chapter 5**, the experiments and the results of the proposed model will be presented along with an error analysis of the results.

2. Background

We give the essential background knowledge for the ease of the reader in two sections: Linguistic background and deep learning background.

2.1. Linguistic Background

In this section, we review background knowledge related to linguistics, especially the main concepts in dependency grammar.

2.1.1. Dependency Grammar

The dependency of a word in a sentence refers to the word's relations to other words in a sentence. For example, in the following sentence:

Thank you, Mr. Poettering .

the word *you* and the phrase *Mr. Poettering* correspond to the same subject and, both of them are affected by the *thanking* action. Therefore, the *head* of the word *you* is *thank* because the word *you* is affected from the action. The same applies to also *Mr.*. We can also describe a similar relation between the words *Mr* and *Poettering*. In this phrase, the name's head is the title. There is punctuation which could be removed from the sentence but normally punctuation helps to build the grammatical relations between words in a sentence. For example, in the example sentence, comma is a bridge between the *thank*ing action and the object *Mr. Poettering*, while the fullstop finalizes the sentence and therefore finalizes the $thank$ing action. So, both comma and fullstop have $thank$ as their head. Each of these head-dependent relationships are called as dependency and discovering the dependencies in a sentence is called dependency parsing in the literature.

We can visualize these relations with a dependency graph as given in Figure 2.1..

We will describe the terms $head$ and $dependent$ in more detail in the next sub-section, but before that, we should look at the history of dependency parsing. Finding the dependencies in a sentence is not a new task in the field. In computational linguistics, even the very first

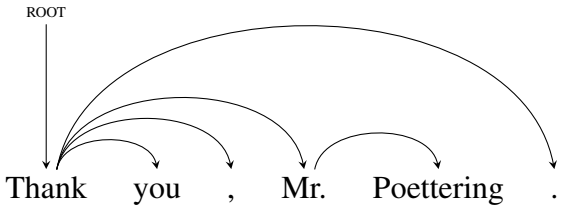


Figure 2.1. An example of dependency parsing for the sentence *"Thank you, Mr. Poettering."*

studies are led by Hays[20] in 1960s. The first modern parser examples were introduced in mid-1900s by Lucien Tesnière [21], a French polyglot. However, the literature shows that the dependency parsing goes back to the 12century that introduces the Arabic parsing, 5century that introduces Panini's Sanskrit parsing, and even goes back to ancient India [22].

2.1.2. Dependency Parsing

Dependency parsing is a task that finds the lexical dependencies between words in a sentence, and thereby extracts the grammatical structure of a sentence. This task can be visualized by dependency trees as given in 2.1..

Dependency is a head-dependent relation between the words. The $head$ is the one that affects the $dependent$s, and in other words each $dependent$ is affected by its $head$. The arrows in dependency trees are always from the head to the dependents.

There are two main approaches applied to dependency parsing problem in the literature: transition-based and graph-based. These approaches will be explained in Chapter 3..

Transition-based approaches are generally based on transition commands and a two-stack structure that contains a dependency stack and a word buffer. Word buffer contains the words in a sentence. Words are drawn from the word buffer and pushed into the dependency stack. If there is a transition between the top two words of the dependency stack, then a dependency is created between them and this operation continues until there are no words in the dependency stack. The last word in the dependency stack would be the $ROOT$, which is the root of the dependency tree; starting point of the whole dependency parsing process.

Graph-based approaches are generally based on performing the entire parsing process as graph operations where the nodes in the graph represent the words in a sentence. For the

sentence, *"John saw Mary"*, imagine a weighted graph G with four vertices where each of them refers to a word including the $ROOT$. Edges store the dependency scores between the words. The main idea here is to find the maximum spanning tree of this graph G. An example is given in Figure 2.2..

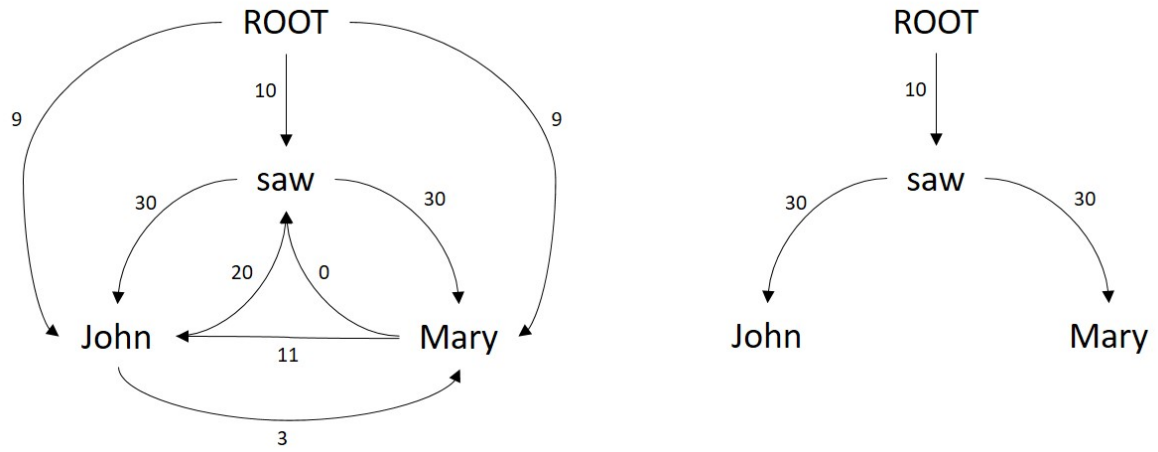


Figure 2.2. Graph-Based Dependency Parsing

So, we can state that the dependencies are between $ROOT$ and saw, saw and $John$; and saw and $Mary$ where the first ones are the heads and the second ones are the dependents.

When the parsing structure is represented as a graph, finding dependencies becomes easier to visualize, and the task becomes finding the highest scored tree. Edge scores in the graph represent the dependency measure between a word couple. It could be defined as a probability or any other measure, which is learned through a mathematical model.

The parsing, no matter which approach is used, creates a dependency tree or a graph, as we mentioned above. There are some formal conditions of this graph:

- Graph should be connected.
  - Each word must have a head.
- Graph must be acyclic.
  - If there are dependencies such as $w1 \rightarrow w2$ and $w2 \rightarrow w3$, there must not be a dependency such as $w3 \rightarrow w1$.
- Each of the vertices must have one incoming edge.

- Each word must only have one head. A graph that includes $w1 \rightarrow w2$ and $w3 \rightarrow w2$ is not allowed in a dependency graph.

- Graph should be projective. (Nevertheless, sometimes both projective graphs and non-projective graphs are accepted.)

  - For example, if there is a dependency $w1 \rightarrow w3$; there should not be a dependency such as $w4 \rightarrow w2$. Dependencies in the dependency tree should not intersect with other dependencies.

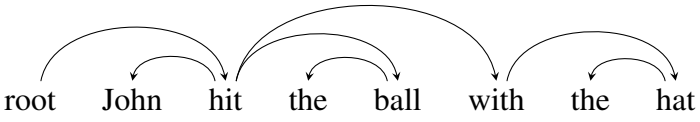


Figure 2.3. A projective tree with none crossing edges

A dependency tree is projective if it can be drawn on the plane with no crossing edges. Figure 2.3. shows a projective dependency graph with no crossing edges and Figure 2.4. shows a non-projective dependency graph with crossing edges. Some algorithms create projective trees while others create non-projectives. For example, algorithms like Chu-Liu/Edmonds [23] or Eisner's algorithm [24] are examples to graph-based dependency parsing and Eisner's algorithm tends to create projective trees while the Chu-Liu/Edmonds algorithm creates non-projective trees.

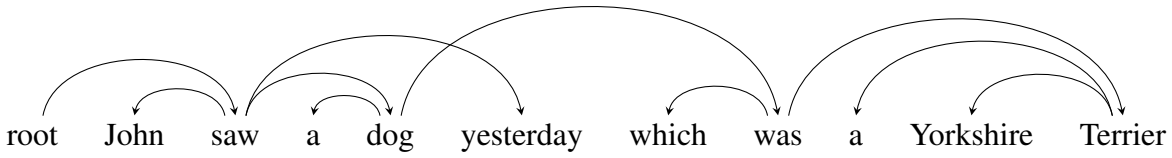


Figure 2.4. Non-projective tree that contains crossing edges

2.2. Deep Learning Background

Deep learning is a machine learning field that is built on Artificial Neural Networks (ANNs) with multiple layers that are capable of extracting features from raw input. ANNs are inspired by biological neurons and brain. They could be considered as the prototypes of the brain,

where the neurons are the activation functions and they are able to learn some specific tasks with the help of these neurons.

There are various deep learning architectures that are based on neural networks. The ones that are related to this thesis are explained in the following sub-sections.

2.2.1. Sequence-to-sequence Models

Neural sequence-to-sequence (seq2seq) models are introduced by Sutskever et al. [25]. The main idea is based on mapping a fixed-length input (input sequence) to a fixed-length output (output sequence). The lengths of the two sequences could be different. It was first applied to machine translation, where the input sequence corresponds to a sentence in one language (source language) and it is mapped to the translation of the sentence in another language (target language). Seq2seq models are applied to various tasks such as speech recognition (e.g. Prabhavalkar et al. (2017) [26]), text summarization (e.g. Dong et al. (2019) [27]) etc.

Seq2seq models are generally composed of two components: encoder and decoder.

Encoder is a structure that encodes the given input sequence and outputs the encoded input sequence. It collects any features in the input sequence that are peculiar to a specific task. Generally, neural network based structures such as Recurrent Neural Networks (RNNs), Long-Short Term Memory Networks (LSTMs) or Bi-directional LSTMS (Bi-LSTMS) are used for learning the features of the input sequence. Thus, the output of the encoder contains valuable information about the input sequence.

Decoder is a structure that decodes the encoded information and generates the output sequence. The output of the decoder is a matrix or probability distribution for prediction. For example, the output of a decoder in a machine translation task can correspond to the probabilities of generating each word in each time step in the output sequence. Decoders generally cannot associate two distant words in the input sequence. So, when the size of the input sequence increases; the accuracy of the predictions in the seq2seq models tend to decrease[4].

Attention mechanisms are found to be useful for associating words in an input sequence while decoding each word into a target sequence. The attention mechanism is based on a cognitive perspective, that claims humans attend to different parts of a text while reading it. Analogously, in a seq2seq model, the decoder is forced to attend to different parts of the

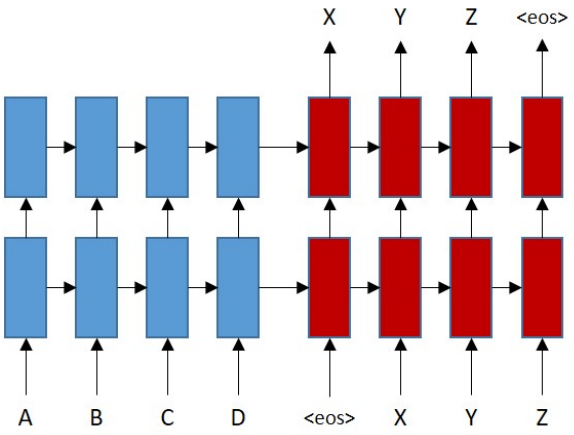


Figure 2.5. An example Seq2Seq Model

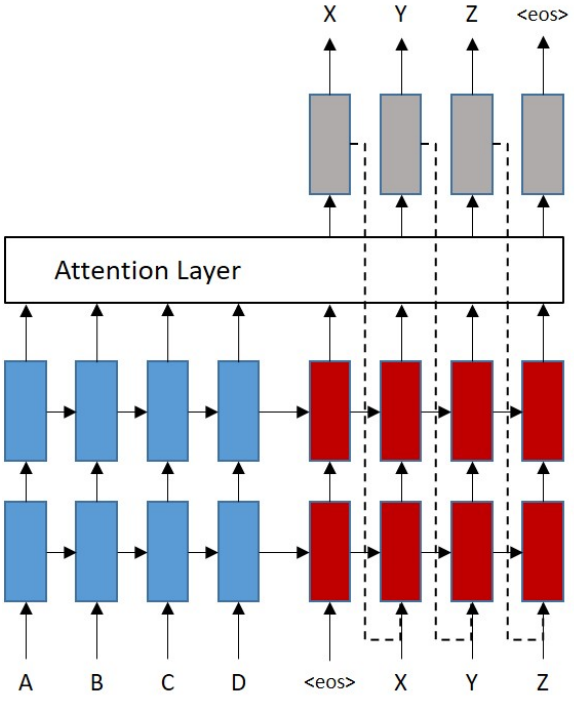


Figure 2.6. An example Seq2Seq Model with Attention

input sequence while generating an item in each time step. The attention mechanism brings one important advantage: the model can handle long distance relations between two further words in the input sequence and this advantage makes the seq2seq models stronger compared to simple seq2seq models without an attention mechanism.

Attention layer is usually a feed-forward neural network that is trained with all other components of the model. Assume that there is a model like in Figure 2.7. which has input

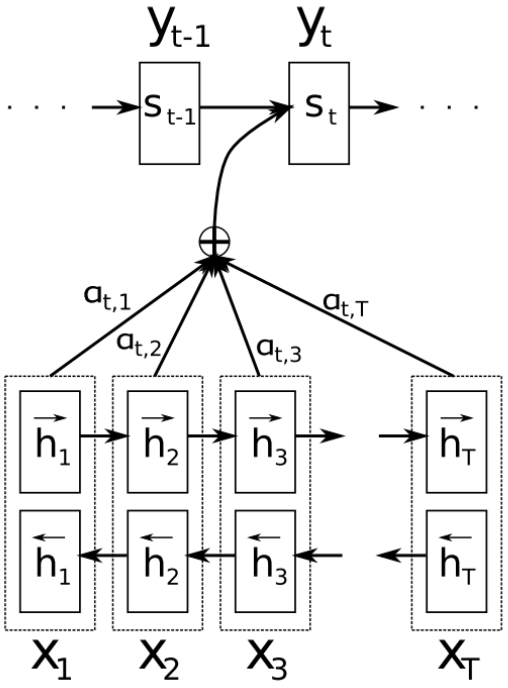


Figure 2.7. Closer look to the Attention Mechanism

embeddings $x = X_1, \ldots, X_T$ and encoder outputs $h = h_1, \ldots, h_T$ and a decoder layer s. The arrows labelled with $a_{i,j}$ are the attention layer and those are the attention weights in the model. Summation of the attention scores will give the *context vector*, which is used to predict the following word. The final score of the encoder and the attention layers are calculated as follows:

$$e_{i,j} = a(s_{i-1}, h_j) \tag{1}$$

where e is the final score, a is the attention layer, s_{i_1} is the output of last decoder, h is the encoded input, i is the time step and j is the counter that iterates over the input embeddings.

After applying softmax to those scores; context vector (c_i) is calculated as a weighted sum as follows:

$$c_i = \sum_{i=1}^{T} e_{i,j} h_j \tag{2}$$

2.2.2. Transformer Networks

Transformer networks are firstly introduced by Vaswani et. al. [8]. In the seq2seq models, encoder and decoder components contain neural network models to learn the input sequence and predict the matching output sequence. However, the authors of the paper reveal that the attention mechanism can be used instead of these recurrent neural networks, which tells everything about the title of the original paper *"Attention is All You Need"*. This reduces the cost of the model and it eases the learning of the model.

Transformer networks are effectively applied to sequential data, and therefore it could be part of a seq2seq model. Transformer networks in a seq2seq model have encoder/decoder stacks which contain one or more encoder(s)/decoder(s). The encoder and decoder stacks contain variations of the specialized versions of Self Attention which is an attention mechanism that mentioned in the paper.Details of the transformer model will be explained in Section 4.2..

2.2.3. Pointer Network

Pointer Network is a sequence-to-sequence model that uses a sequence of pointers to the items of input series instead of converting one sequence into another. The model is firstly proposed by **Vinyals et al. (2015)** [1] and is generally used for ordering the items of a variable-length sequence or set.

As we can see from the left part of Figure 2.8., while an LSTM or RNN-based model processes the input sequence (blue ones), the dimensionality of the output sequence (purple ones) is determined by the dimensionality of the problem and it remains the same during training and inference [25]. However, on the right hand side of the figure, we can see that every output of the output sequence points to each of the input nodes. So, this model can handle the dimensionality problem with this simple approach.

This approach is used in the model that is proposed in this thesis. The output of the Encoder stack is an $n \times dim$ sized matrix where n is the number of words in the sentence and dim is the dimension of vectors in the model. However, we need an $n \times n$ matrix for Eisner's algorithm[24] because it parses the sentence and needs a score matrix which contains each word's relation to all words in a given sentence. Thus, with Pointer Networks, we can transform the given input to a squared matrix without losing any relational information.

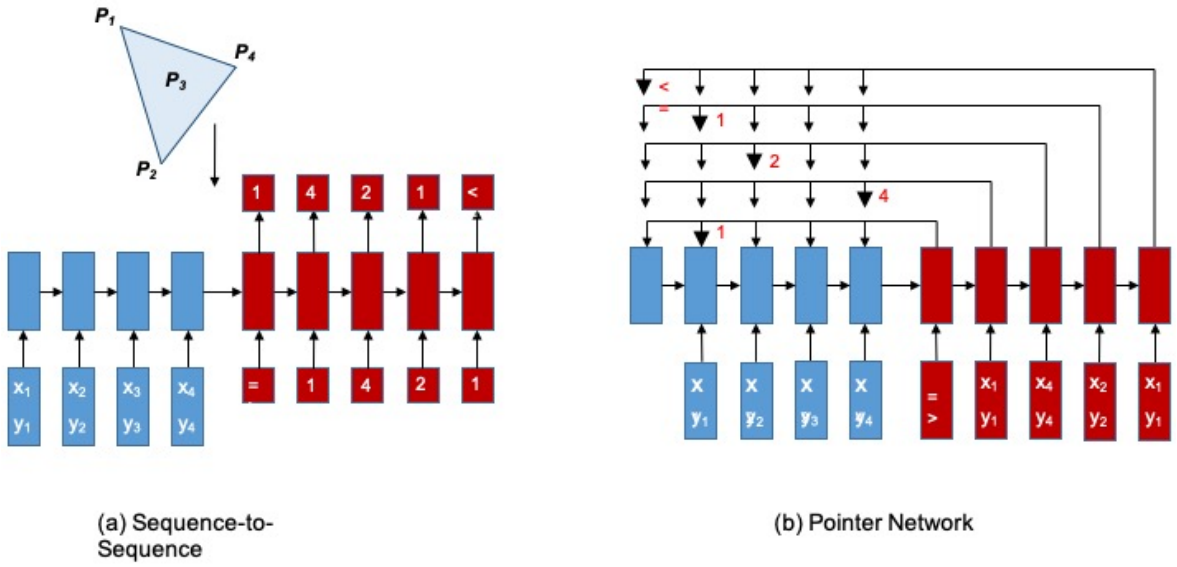


Figure 2.8. a) Basic seq2seq model and b) Pointer Network [1]

2.3. Learning and Neural Networks

Some concepts related to learning in neural networks are covered below.

2.3.1. Loss Function

Loss function is a simple mathematical function that evaluates how well the prediction of the model fits to the gold value. Weights that are learned from the model are updated after calculating the loss function. This update operation is performed by back-propagation and an optimizer in neural networks. An example neural network is given in Figure 2.9..

Learning means updating weights in each iteration (i.e. epoch) until the loss gets close to zero. Learning can be divided into two parts: forward pass and backward pass. At forward pass, scores are calculated for each node. For example;

$$h1 = F_{act}(l1 \times w1 + l2 \times w2) \tag{3}$$

where F_{act} is the activation function such as sigmoid, tanh, etc.

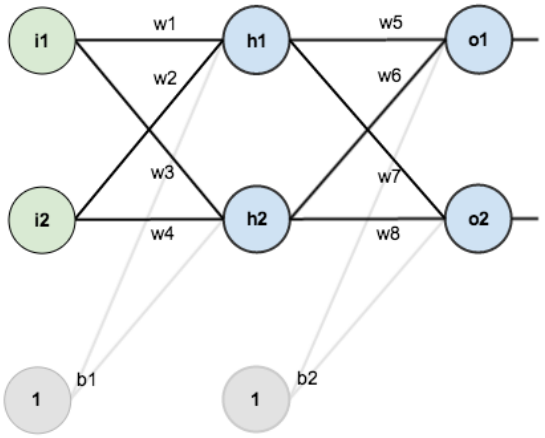


Figure 2.9. A simple Neural Network

After calculating all scores in the first layer, we can predict the output scores as given below:

$$o1 = F_{act}(h1 \times w5 + h2 \times w6) \tag{4}$$

After a sufficient number of epochs, the prediction begins to fit onto the gold values. Different loss functions can be applied depending on the task. For example, while the Mean Square Error (MSE) loss function is used for regression tasks, Negative Log-Likelihood loss function is generally used for classification tasks.

Since our example in Figure 2.9. is a simple model, we can use Squared Error Function:

$$E_{total} = \sum \frac{1}{2}(target - output)^2 \tag{5}$$

Once the errors are summed over for each output node (i.e. o1 and o2), weights are updated by back-propagating the errors. For this purpose, gradients based on each weight are calculated to determine how much each weight will be changed in that epoch. For example, in order to update w_5, we need to calculate the derivative of that weight according to the total loss value: $\frac{\delta E_{total}}{\delta w_5}$.

Derivatives are calculated by using chain rule as follows:

$$\frac{\delta E_{total}}{\delta w_5} = \frac{\delta E_{total}}{\delta o_1} \times \frac{\delta o_1}{\delta net_{o1}} \times \frac{\delta net_{o1}}{\delta w_5} \tag{6}$$

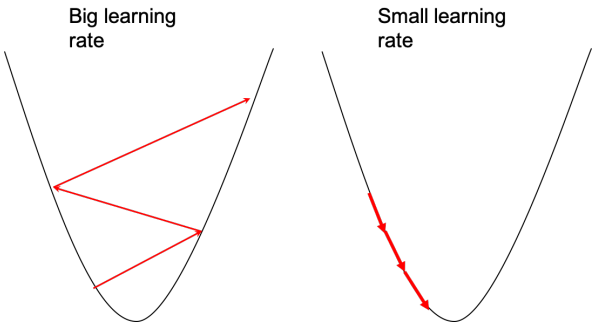


Figure 2.10. The effect of learning rate on the loss function

where net_{o1} is the input of F_{act} in equation 4. And the new value for w_5 will be:

$$w_5^+ = w_5 - \alpha \times \frac{\delta E_{total}}{\delta w_5} \tag{7}$$

where α is the learning rate.

After updating all weights in the network, the forward pass and backward pass are applied again and again until the loss becomes close enough to zero or any value under a pre-defined threshold value, or till a pre-defined maximum number of iterations is exceeded.

2.3.2. Optimizer

Optimizer is an approach used for updating weights in a model based on the calculated loss. So, an optimizer is run after the loss is back-propagated.

Learning rate is a ratio that is used to update the weights. Imagine a parabola $y = x^2$. Our starting point is the leftmost point of it and our aim is to reach the local minimum. If the learning rate is large, the loss will change unorderly. It will probably go to the right and left and right on different epochs. However, if the learning rate is small enough, our loss will follow the curve and reach the local minimum efficiently.

In Figure 2.10., the parabola represents the change of the calculated loss according to the current weights, w. In other words, it shows how the α parameter effects the prediction of w.

The aim is to reach the local minimum at the parabola, where the value of the loss function is the smallest. Red dots show the calculated loss values in different iterations and the red

lines show the change of the loss values from one iteration to the another. If the learning rate is large, our predictions will be less effective. However, if the learning rate is small enough, we can reach the local minimum more efficiently.

Another important concept in an optimizer is **regularization**. It is a penalty constant that is added to the loss function to penalize larger weights. Weights are updated during training but they should not memorize the training set. If memorization happens, the model would perform very well on training data but it will perform poorly on real-world examples, on a test dataset. This issue is called as **overfitting** and regularization is a way to reduce the risk of overfitting.

Gradient Descent[28] is the ancestor of all optimizers, which is used to reach the local minimum. There are different types of optimizers such as Adam [29], Adagrad [30], etc. Adam optimizer differs from the other optimizers with its intuition on the concept of momentum by injecting values from previous gradients to the current gradient. While the Adam implements the exponential average of the gradients to scale the learning rate, Adagrad takes the simple average of them. However, in case of the gradient descent, it uses same learning rate for each parameter and updates all weights at once.

3. RELATED WORK

3.1. Introduction

As aforementioned, Dependency Parsing is performed by mainly two approaches: Transition-Based and Graph-Based. Related work that follows these approaches will be reviewed in this chapter. The categorization will be based on these two approaches. Moreover, the ones that are experimented on Turkish will be further discussed. Turkish is a language that the researchers are shy of working on it because it is an agglutinative and low-resource language. However, the work in this thesis mainly focuses on Turkish language. So, we first review the dependency parsing literature based solely on Turkish language.

3.2. Related Work on Turkish Dependency Parsing

Turkish is an agglutinative language that builds the words with countless suffixes that are attached to the end of the word. When talking about Turkish, a typical example would be:

Çekoslovakyalılaştıramadıklarımızdanmışsınızcasına
(Like you are the one from those who we couldn't make Czechoslovakian)

This example is generally given for the Morphological Segmentation task. However, because of the number of affixes, such inflected words become sparse in a corpus. Thus, finding a word's dependency gets harder because of sparsity and out-of-vocabulary (OOV) problem.

Eryigit and Oflazer (2006) [31] come with the idea of using inflectional groups (IGs) for dependency parsing. In their study, the authors use a statistical parser that firstly computes unit-unit relations where the units are words or IGs and then finds the maximum spanning tree from these computed relations. They have three baseline models: Word-based, IG-Based, and IG-Based with word-final IG contexts which is an IG-Based model with strict outputs. As expected, IG-Based models give the best results.

Eryigit, Oflazer and Nivre (2008) [2] show that the morphological structure plays a crucial role in Turkish dependency parsing. The authors show that parsing a sentence considering

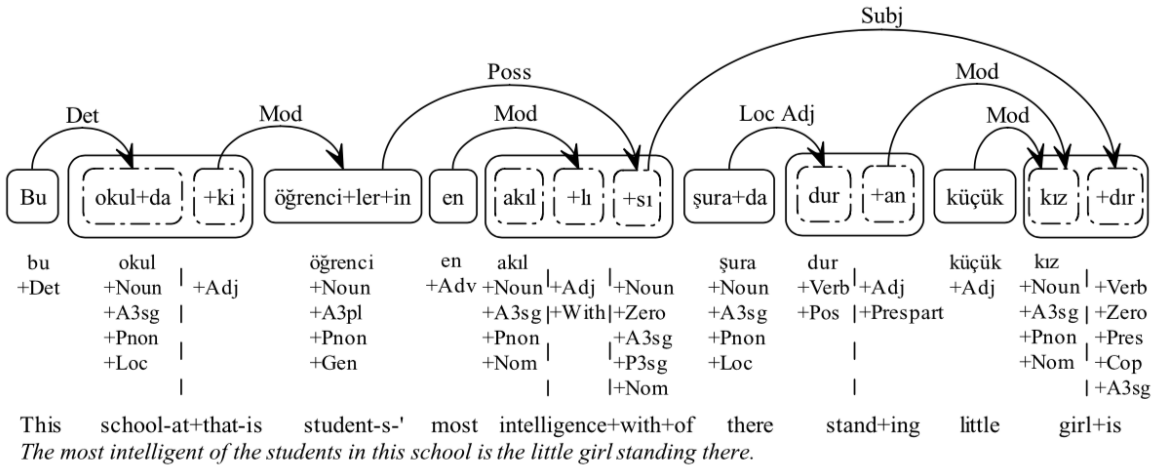


Figure 3.1. Inflection Groups (IGs) used in dependency parsing [2]

the IGs, which are sublexical units of a word, outperforms dependency parsing based on word tokens of sentences.

An example dependency graph from Eryigit, Oflazer and Nivre (2008) [2] is given in Figure 3.1.. Here, the rectangles correspond to the inflection groups. Finding the dependencies between these inflectional groups provides an improvement on the performance of the dependency parsing. In this work, two different models are introduced. A probabilistic model with beam search algorithm and a rule-based parser based on **Nivre et al. (2006)** [32], which uses Support Vector Machine (SVM) classifier that evolves to MaltParser (**Nivre et al. (2007)** [33]) are used to predict the next dependency relation in a sentence. The two models are used to parse the sentence with a Word-based approach and IG-based approach. Their findings show that the IG-based models outperform Word-based models and using IGs increases the accuracy in both probabilistic and rule-based models. They claim that the rule-based model is more accurate than the probabilistic model. Moreover, that study shows that the morphological structure contains *"important syntactic information"* in Turkish.

Oflazer (2014) [3] analyzes different NLP tasks on Turkish. In the dependency parsing task, the author underlines the importance of IGs and morphological units in dependency parsing. Also, he explains the ambiguity and variety of Turkish sentences with the example in Figure 3.2.

As we can seen from Figure 3.2., the order of the words in a sentence brings different meanings. This issue affects the semantics mostly, however, it also shows the importance of the

- Ekin Çağla'yı gördü. *(Ekin saw Çağla.)*
- Çağla'yı Ekin gördü. *(It was Ekin who saw Çağla.)*
- Gördü Ekin Çağla'yı. *(Ekin saw Çağla (but was not really supposed to see her.))*
- Gördü Çağla'yı Ekin. *(Ekin saw Çağla (and I was expecting that.))*
- Ekin gördü Çağla'yı. *(It was Ekin who saw Çağla (but someone else could also have seen her.))*
- Çağla'yı gördü Ekin. *(Ekin saw Çağla (but he could have seen someone else.))*

Figure 3.2. Ambiguity in Turkish grammar [3]

syntactic structure. **Eryigit (2012)** [34] makes an analysis on parsing in raw datasets in Turkish and shows that the locations of words in a sentence plays a crucial role in parsing.

Cakici and Baldridge (2006) [35] also use IGs in their work. Their study shows the tendency of the structure of the dependency trees in Turkish language as either being projective or non-projective. They employ IG based dependency parsing with two main parsing models: head-driven generative parsing and discriminative dependency parsing. Head-driven generative parsing adopts Collins' algorithm (Collins (1997) [36]) to create non-projective trees. Discriminative dependency parsing model uses two algorithms; Eisner's algorithm (Eisner (1996) [24]) to produce projective trees and McDonald et al. (2005) [37] to produce fully non-projective trees. The study shows that the discriminative parsing algorithms outperform phrase-structure head-driven parsers. Moreover, the paper shows that the non-projective trees give slightly better accuracies than the projective ones like Eisner's algorithm for Turkish.

Another study that shows the relation between dependency parsing and IGs is presented by **Cetinoglu and Kuhn (2013)** [38]. In this study, morphological segmentation and parsing are learned jointly. The study also shows that using IG-based parsing increases the accuracy. They perform several experiments to include morphological features: not using them, using them only at training and using them at both training and parsing. The latter one gives the best results.

Yuret (1998) [39] addresses the dependency's quality between two distant words in longer sentences. The author builds a *bootstrap acquisition* method and calculates a probability called *lexical attraction*. Also, this model contains a simple memory unit, which basically contains the counts of relations between two words. Thus, the idea is that if two words are located together in the dataset, it is more likely to have a dependency relation between them.

Gonzalez and Rodriguez (2018) [40] introduce a parser based on the Covington's parser [41] with only one difference: instead of the *No-Arc* relation in the regular Covington model, they adapt *non-local* principle. In this principle, instead of just saying there is No-Arc; they try to parse No-Arc with *Left-Arc* (LA) and *Right-Arc* (RA) transitions. In Covington's study, there are two stacks λ_1 and λ_2 and if the Right-Arc transition comes, word is shifted from λ_1 to λ_2 and vice versa for Left-Arc transition. If there is No-Arc, they shift the latest word from λ_1 to λ_2. In Gonzalez and Rodriguez (2018) [40], they use a parameter k for LA and RA instead of a No-Arc transition in order to define how many words will be shifted from one stack to other. So, RA_k means that shift k leftmost words from λ_1 to λ_2 because words before the k word has no-arc with any other word in the sentence. They use a parameter which stores the count of the dependency arcs in the sentence. Therefore, the authors introduce a novel transition-based non-projective parser called **NL-Covington** which outperforms other transition-based parsers on Penn Treebank dataset (UAS = 94.5% and LAS = 92.4%) and improves the Covington's accuracy on other languages such as Turkish, Spanish, Swedish, etc. in CONLL datasets. Their accuracy is 81.30% UAS and 71.28% LAS for Turkish while the Covington is 80.29% and 70.68% respectively.

McDonald et al. (2006) [42] propose a two-stage parser which firstly parses the sentence with a standard parsing algorithm and then labels the parsing with a dynamic algorithm such as Viterbi [43]. This work has 74.7% UAS and 63.2% LAS accuracy on Turkish dataset in CONLL-X shared task [44].

Another work is presented by **Titov and Henderson (2007a)** [45]. In this work, the authors use Incremental Sigmoid Belief Networks (ISBN)[46] which is also proposed by Titov and Henderson (2007b) [46]. ISBN is an improved version of Sigmoid Belief Networks (SBN)[47] and it is similar to Hidden Markov Models (HMMs) because of using binary latent variable vectors to encode information about the parsing history. However, instead of using dependency edges between states like in HMM; they use dependency edges between latent variables. They also employ a Beam Search-like heuristic search algorithm that is described in another study of the authors [48]. In that paper, they obtain 86.2% UAS and 79.8% LAS on Turkish dataset in CONLL-2017 shared task [49].

In addition to these probabilistic models, there are numerous neural network based models that are experimented on Turkish. One of them is by **Akdemir and Gungor (2019)** [50], which is a recent work that jointly learns the dependency trees and named entity tags. Their model is based on an encoder that is built upon Long-Short Term Memory Networks

(LSTMs) and Multi-Layer Perceptrons (MLPs) and a decoder that uses Conditional Random Fields (CRFs) [51] at decoding.

In the study of Akdemir and Gungor (2019) [50], they propose three different models (two of them contain dependency parsing and the other one only learns NER features). The one that gives the best score (60.0% average LAS-UAS accuracy on Turkish dataset in CONLL-2018 shared task [52]) is similar to the joint parser proposed by **Nguyen and Verspoor (2018)** [53].

The proposed model in **Nguyen and Verspoor (2018)** [53] jointly learns POS tags and dependencies using the POS tags. Their model takes input word vectors as a concatenation of the word and character embeddings. After learning POS tags from these embeddings using a BiLSTM and MLP-based tagging component they use the input vectors and learned POS embeddings in order to learn dependencies in a layerered structure. Their parsing component concatenates the input embeddings and the POS embeddings and gives them to BiLSTM model for encoding sentences. After encoding, they create score matrices through MLP and gives that score matrix to the Eisner's algorithm in order to find the highest scoring projective parse tree. Their model has an accuracy of 70.53% UAS and 62.55% LAS in Turkish IMST dataset in CONLL-2018 shared task [52].

Kondratyuk and Straka (2019) [54] presents a parser called UDify that uses multilingual BERT [55] self-attention features to encode sentences. Their model outperforms the transition-based model called UDPipe by **Straka et al. (2018)** [56] which generates transitions in a search-based oracle. UDify model in Kondratyuk and Straka (2019) gives 74.56% UAS and 67.44% LAS for Turkish IMST dataset and 67.68% UAS and 46.07% LAS for Turkish PUD dataset in CONLL-2018 shared task [52].

In addition to the models that use BERT, there are plenty of works that use deep-contextualized vectors like ELMo [57] such as **Che and Liu (2018)** [58] and **Kulmizev and Nivre (2018)** [59]. The first study [58] uses the model introduced by Dozat and Manning (2016) [5] as the baseline model and adds the ELMo [57] embeddings to it and performs 66.44% LAS accuracy for Turkish IMST dataset in CONLL 2018 shared task [52]

The latter paper[59] shows the effect of BERT [55] and ELMo [57] embeddings on transition-based and graph-based parsers. According to their study, using ELMo or BERT increases the accuracy but the BERT gives 1% more accuracy and achieves 64.9% LAS score in Turkish IMST dataset in CONLL 2018 shared task [52]

Ballesteros et al. (2015) [60] show that using character-based embeddings increases the accuracy when compared to word-based embeddings. Especially the languages like English have some defined affixes like $-s$ or $-ing$. So, instead of the word itself, if we extract the probability of words that ends with ing; their lexical heads will be similar words too. However, in Turkish, there are plenty of affixes so even if this approach increases the accuracy for one dataset, it may fail for another dataset. This study uses a stack-based LSTM network and obtains 76.32% UAS and 64.34% LAS for Turkish IMST dataset in CONLL-2018 shared task [52].

Another stack-based LSTM approach is introduced by **Ma et al. (2018)** [9], where the authors suggest two models for the parsing task: One of them is called $STACKPTR$ which is a stack-based LSTM model that uses BiLSTM-CNN network in encoding and LSTM in decoding, and the other one is called $BiAff$, an improved version of Dozat and Manning (2016) [5] (slight changes like dropout value and adding one perceptron to the model's attention). Their $BiAff$ model gives 79.84% UAS and 68.63% LAS which are the best results in Turkish currently for Turkish dataset in CONLL-2006 shared task [44].

Besides these models, some of the studies such as Turk et al. (2019) [61], Ambaty et al. (2012) [62] and Sulubacak et al. (2016) [63] present results on Turkish datasets as well.

To conclude, Turkish is a language that is hard to deal with. There are no recent papers that mainly focus on Turkish dependency parsing. However, the papers that use CONLL datasets generally test their models on Turkish CONLL datasets too.

We will discuss the related work on graph-based and transition-based parsers in the following sections which are introduced for other languages and not solely for Turkish language.

3.3. Literature Review on Graph-Based Dependency Parsing Models

Graph-Based Dependency Parsing is a task that firstly represents the sentence as a graph and then finds the maximum spanning tree in this graph in order to find the correct dependency tree. In general, there is a distinction between the Graph-Based parsers and Transition-Based parsers. However, some studies do not define their parsers as graph-based or transition-based. Because of this reason, the ones that do not contain transition rules or stacks are classified as Graph-Based parsers in this thesis.

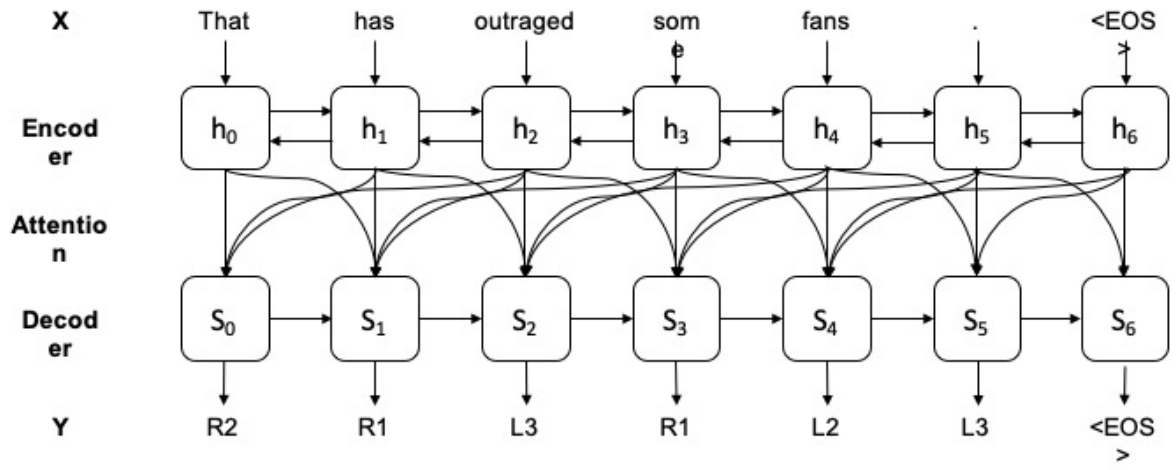


Figure 3.3. The Seq2Seq Parser introduced by Li et al. (2018) [4]

There are especially two main studies that can be counted as the *baseline* neural models. One of them is presented by **Li et al. (2018)** [4] and it is a typical sequence-to-sequence (seq2seq) model that uses a bi-directional LSTM in its encoder and Long-Short Term Memory (LSTM) in its decoder. Between the encoder and decoder stages; there is an attention mechanism to encode the word's context as can seen in Figure 3.3..

With the attention mechanism and a sub-root decomposition & beam search algorithm to create dependency trees, they aim to increase the parsing accuracy in longer sentences. They obtain 94.11% UAS on Penn Treebank and 88.78% UAS on Chinese Treebank.

Kiperwasser and Goldberg (2016a) [64] propose a model based on two BiLSTMs and a MLP models, where one of them is for transition-based parsing while the other one is for graph-based dependency parsing. **Kiperwasser and Goldberg (2016b)** [65] propose another BiLSTM based model which contains two RNNs for finding right and left arcs in the dependency tree.

Dozat and Manning (2016) [5] propose a novel parser that uses a biaffine attention mechanism. The model is a modified version of th one presented by Kiperwasser Goldberg (2016 [64]), Hashimoto et al. (2016) [66], and Cheng et al. (2016) [67]. They create a model(3.4.) which is built upon a BiLSTM + MLP that uses biaffine attentions instead of using bilinear or MLP mechanisms. Their deep biaffine attention uses the recurrent states directly and can be defined mathematically as follows:

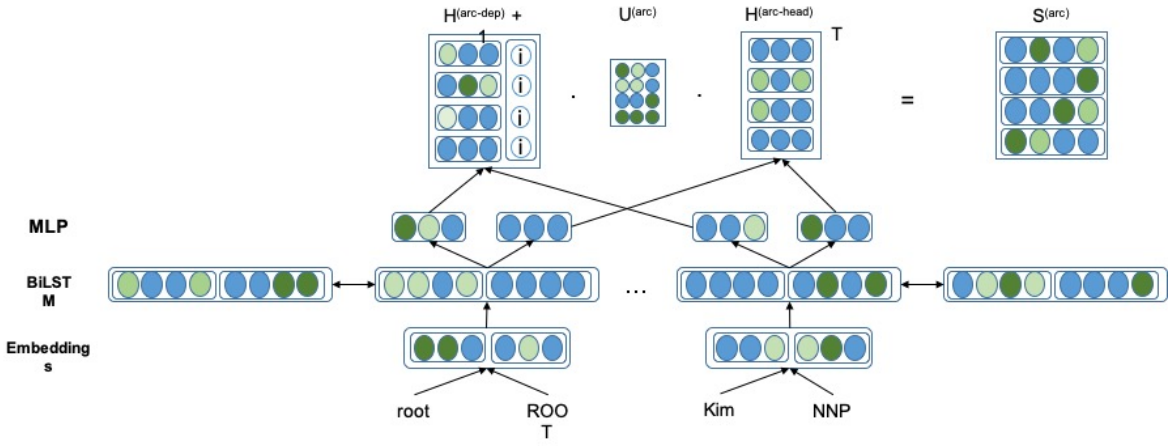


Figure 3.4. The model introduced by Dozat and Manning (2016) [5]

$$\mathbf{h}_i^{arc\_dep} = MLP^{arc\_dep}(r_i) \tag{8}$$

$$\mathbf{h}_j^{arc\_head} = MLP^{arc\_head}(r_j) \tag{9}$$

$$\mathbf{s}_i^{arc} = H^{arc\_head}U^{(1)}h_i^{arc\_dep} + H^{arc\_head}u^{(2)} \tag{10}$$

where h is an MLP that represents the head and dependency arcs, H is the recurrent units (BiLSTMs) and s is the score matrix.

The model introduced by Dozat and Manning (2016)[5] increases the accuracy of its baseline models. The Biaffine parser that is proposed in Dozat and Manning (2016) is a baseline for many other papers such as **Zhou and Zhao (2019)** [6] and **Li et al. (2019)** [68]. The latter one is composed of a Self-Attention mechanism in the Transformer Model with the deep biaffine network of Dozat and Manning (2016). The first one forms the dependency tree as a Head-Driven Phase Structure Grammar (HPSG)[69].

They present two parsers in their study: One of them is called division-span decoder that is used along with a constituency parser by Kitaev and Klein (2018) [70] that uses Self-Attention mechanism and the other one is called joint-span decoder that uses the parser suggested by Dozat and Manning (2016). Their joint-span model that uses XLNet [71] or BERT [55] in the encoder gives the best scores.

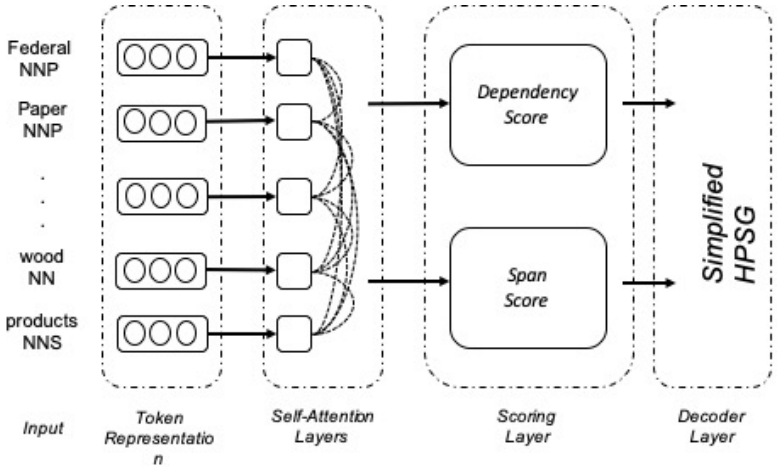


Figure 3.5. The model introduced by Zhou and Zhao (2019) [6]

Mrini et al. (2019) uses the concept of forming the dependencies to Head-Driven Phase Structure Grammar (HPSG)[69]. They use XLNet in the encoding step and proposes a new attention mechanism called Label Attention Layer which is an evolved version of Self-Attention. This work is at the top of the accuracy list between the models that use Penn Treebank as dataset in dependency parsing.

Another LSTM-based model introduced by **Choe and Charniak (2016)** [72] considers the dependency parsing task as part of language modelling (LM) and parses the sentence with a LSTM-LM architecture which generates parse trees simultaneously with the n-gram probabilities.

Another study introduced by Ji et al. (2019) [73] proposes a Graph Neural Network (GNN) that can be counted as an extended version of the biaffine model.

The recent studies generally focus on the encoder of the neural network models because a better encoding of an input with its context eliminates most of the cons of the sequence models. For example, Hewitt and Manning (2019) [74] and Tai et al. (2015) [75] aim to increase the effectiveness of LSTM-based encoders while Clark et al. (2018) [76] develop a new attention based approach for encoding, which is called Cross-View Training (CVT).

As we mentioned above, most of these models are trained and tested on Penn Treebank. There are some works which just concentrate on Universal Dependencies and we can count Stanford's CONLL parser (Qi et al. (2019)) [77] as one of them.

Moreover, there are a few unsupervised approaches applied for dependency parsing task. The *iterative re-ranking* model of **Le and Zuidema (2015)** [78] is the one that outperforms other unsupervised models in dependency parsing task for English Penn Treebank WSJ corpus [79].

Spitkowski et al. (2013) [80] propose that the parsing operation can be brute-forcely performed. They proposed adding two additional operations (Transforms (Unary) and Joins (Binary)) to the probabilistic model during training to transform words in order to reach the local optima easier.

Klein and Manning (2004) [81] propose an unsupervised model that combines DMV (dependency model with valence) and CCM (constituent context model) models and creates dependency trees with the Expectation-Maximization (EM) principle. Their model is experimented on English, German and Chinese and gives high scores compared to other unsupervised models.

Another unsupervised model is presented by **Smith and Eisner (2005)** [82] which represents sentences as lattices and perform different modifications on the sentence in order to increase the number of input sentences and improve the probability of an original sentence. They propose a method called Contrastive Estimation (CE) which is a principle that improves the original sentence's probability by decreasing the modified false sentence's probability.

Headden et al. (2009) [83] combine the latter two studies (Klein and Manning (2004) [81] and Smith and Eisner (2005) [82]) and shows that the scores substantially increase with the use of Contrastive Estimation along with Expectation Maximization algorithm.

In addition to the unsupervised models, **McDonald et al. (2005)** [37] show the effect of Maximum Spanning Tree (MST) algorithms and the Eisner's parser **[24]** on the languages that contains non-projective dependencies. Eisner's parser is an improved version of Cocke–Younger–Kasami algorithm (CYK algorithm)[84][85][86] and it is a competitor to Chu-Liu/Edmonds[23]. It is one of the simplest parsing algorithms and is currently used as a baseline parser in many dependency parsing frameworks.

The model that is proposed in this thesis is an example of Graph-Based Dependency Parsing. However, knowing some of the transition-based dependency parsers will help understanding the choice of graph-based parsing in this thesis. Following section reviews the related work on transition-based dependency parsing.

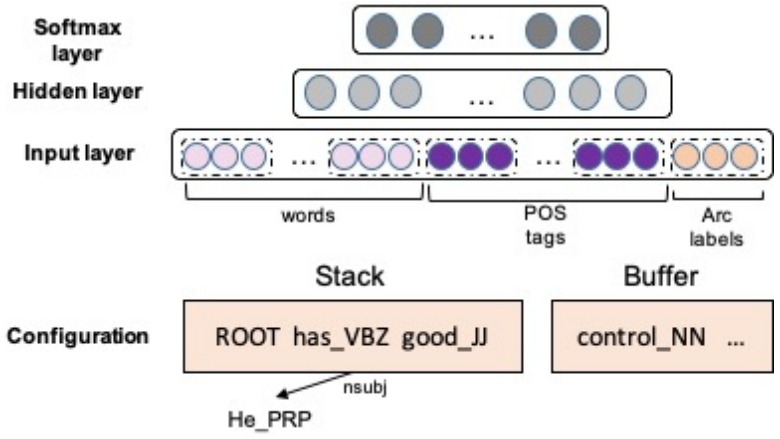


Figure 3.6. The neural parser presented by Paper Chen and Manning (2014) [7]

3.4. Literature Review on Transition-Based Dependency Parsing Models

Transition-Based Dependency Parsing is an approach to find dependency trees with, generally, two stack structures (input buffer and parsing stack) by considering a set of transitions. Usually, the input buffer, an ordered list of words in a sentence, feeds the parser. For each word, model checks the parsing stack; if there is a dependency relation (which is from the set of transitions) between the current word and the word at the top of the stack, then the parser relates these two words and continues for other words until there is no word in the stack. Final word in the stack will be the $ROOT$.

Chen and Manning (2014) [7] propose a neural network model to perform transition-based dependency parsing. The architecture of the model is given in Figure 3.6..

Chen and Manning (2014) [7] propose a standard Recurrent Neural Network (RNN) model with input embeddings that contain words, Part of Speech (POS) tags and arc labels that are concatenated to build the final input to be fed into the RNN. They propose various modifications to the model by injecting single-word, word-pair and three-word features. The paper is one of the best parsers in its era and outperforms other parsers such as MaltParser (Nivre (2007)) [33] and MST Parser [42] in most languages.

Weiss et al. (2015) [87] follow a similar architecture that follows the model proposed by Chen and Manning (2014) [7]. Weiss et al. (2015) introduce a model that contains an extra hidden layer and different activation functions compared to the model proposed by Chen and Manning. They obtain 94.26% UAS and 92.41% LAS score in Penn Treebank (PTB) dataset.

There are other studies that make use of RNNs. For example, Dyer et al. (2015) [88] and Chen et al. (2015) [89] are also examples to such neural parsers.

Mohammadshahi and Henderson (2019) [90] use Transformer Model for dependency parsing. They inject graph features for the input and output embeddings of the Transformer in order to extract information from the graph for transition-based parsing purposes. Thus, the Graph2Graph model is introduced in their study. When the model is extended with pre-trained BERT embeddings, it outperforms other transition-based models.

Andor et al. (2016) [91] propose a feed-forward neural network model that is used for dependency parsing, PoS tagging and sentence compression tasks. The model adopts Global Normalization instead of standard Local Normalization, so it can handle *the label bias problem* efficiently.

Currently, the model that gives the best results is presented by **Gonzalez and Rodriguez (2019)** [92]. The authors propose a transition-based algorithm that is similar to the $STACKPTR$ model [9]. While the original work uses a standard top-down parsing, Gonzalez and Rodriguez (2019) [92] use left-to-right parsing. In other words, while iterating over the words in a sentence, they parse the sentence into n attachments where n is the number of words in the sentence. Thus, their model does not require a buffer or stack to predict the tree. The authors claim that the model successfully finds and holds the single-head dependencies; however, during the parsing of the sentence, sometimes it may produce cycles. As we mention in Section 2.1.2., there must not be a cycle in the dependency graph. So, here the authors check and fix this issue in an additional step with a complexity of $O(n)$. So, they can predict the final parse trees in $O(n)$ time complexity. Finally, they obtain 96.04% UAS and 94.43% LAS scores in English PTB dataset.

In addition to these models that are described above, there are some works such as greedy parsers of **Ballesteros et al. (2016)** [93] and **Kuncero et al.(2016)** [94] or the high-performance parser *arc-swift* of **Qi and Manning (2017)** [95].

Nivre and McDonald (2008) [96] show that we can integrate graph-based and transition-based parsers by integrating their features. From this point of view, studies such as **Goldberg and Elhadad (2010)** [97], **Spitkovsky et al. (2010)** [98], **Ma et al. (2013)** [99], **Ballesteros and Bohnet (2014)** [100] and **Zhang and Clark (2008)** [101] work on the methods, features and/or on a common model that can be used for both transition and graph-based parsing.

3.5. Conclusion

To conclude, there are different approaches employed in dependency parsing without considering whether it is graph-based or transition-based. As the one can see from the reviewed related work above, recent papers are barely using Turkish datasets for dependency parsing.

The transformer-based models are promising for future works, especially in sequence-to-sequence tasks. Thus, as a motivation from all of these related works, we decided to use a popular model, transformer, on a resource-scarce language, that is Turkish.

Following section contains detailed information about the proposed model in this thesis. Firstly, the baseline model will be mentioned. After that, components and their details will be explained.

4. MODEL

4.1. Introduction

As we mentioned in Chapter 1, there are two approaches applied to dependency parsing, namely transition-based and graph-based parsing. Our proposed model can be categorized as a graph-based model that uses neural networks in a sequential-to-sequential structure.

We use the Transformer network [8] which contains an Encoder-Decoder structure with multi-head attention. We extend the baseline Transformer network architecture with extra steps on decoding and the loss function and thereby obtaining a specialized version of the original model for especially adopting it for parsing dependencies.

4.2. Baseline Transformer Model

The Transformer Model is firstly proposed by Ashish Vaswani and the Google Brain Team in 2018 with the paper "Attention is All You Need" [8]. The model is proposed due to some inadequacies of standard seq2seq models. For example, a standard seq2seq model cannot be paralleled easily and it fails to learn relations between two distant words in a sequence.

As we mention in Section 2.2.1., seq2seq models contain encoder-decoder layers for representing and converting the sequence to another, and one or more attention layers between the encoder and decoder layers. A seq2seq model encodes the input sequence generally with RNNs or LSTM/Bi-LSTMs. A target sequence is generated by the decoder that may also use an extra attention layer to focus at some parts of the encoded sequence in each time step.

This might be an effective method for shorter sentences. Howeve, for longer input sequences, it fails to decode effectively because the model starts to give high scores (i.e. attention weights) to the words that are closer to each other and the long distant relations become impossible to be detected.

Vaswani et al. (2017) [8] realize that using neural network architectures such as RNNs or LSTMs in encoding and decoding is not required. Their study claims that attentions are sufficient to encode and decode each sequence. Thus, the idea of "Self Attention" is proposed.

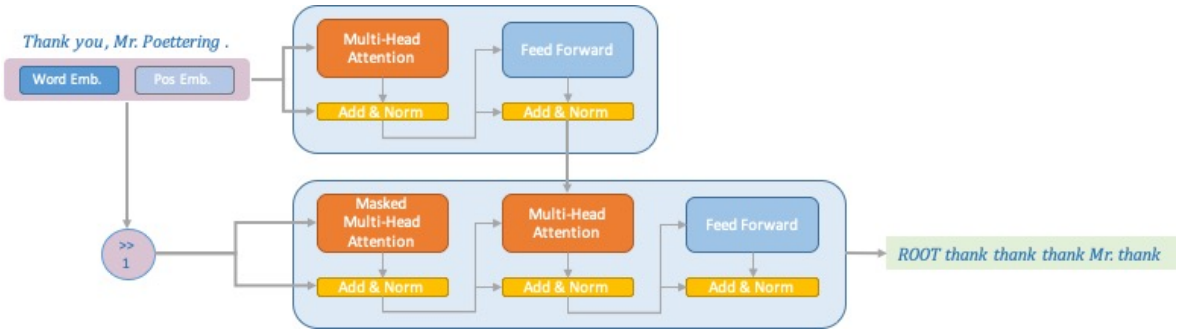


Figure 4.1. The Transformer Network architecture [8].

As Figure 4.1. illustrates, both the encoder (on the left) and the decoder (on the right) stacks contain attentions in addition to Feed Forward neural networks and layer normalization. Details of these structures will be explained in the next section. However, it is good to mention about Query, Key and Value matrices here, which are the key components of a Self Attention mechanism. Query, Key and Value are the matrices which are learned during training and they represent the contextual relations between words. The query is usually the hidden state of the decoder. A key is the hidden state of the encoder and the value is the normalized weight that represents the weights in the encoder. Dot product of the key and the query creates the value. Details of the self-attention is given in Section 4.4..

Positional encoding is used to inject positional information for each encoded word since there is not a sequential structure such as a recurrent neural network in the self attention mechanism. Thus, the relation between closer words and distant words can be defined better.

Layer Normalization is applied after each attention and feed forward component. It is used to normalize the data, for the current sentence. If batch normalization is used, it normalizes the data for the whole batch.

In the transformer network, outputs are generated by shifting the outputs one to the right. In encoder stack, we relate all words in the sentence with other words in the same sentence. However, in the decoder stack, words are predicted based on the ones which are already predicted. For example, if the third word is currently being predicted, then it is assumed that the fourth word is not known yet thereby having a sequence problem. In other words, the third one is predicted based on the first and the second words in the sequence.

The self attention model contains one encoder and decoder stack. Usually, three or four encoder/decoders are contained in each stack. For each stack, each encoder's output is fed

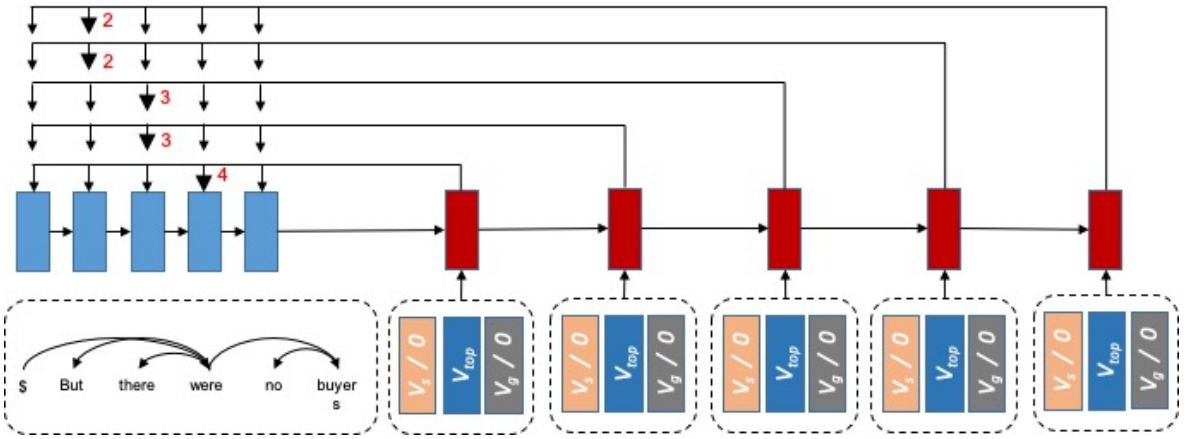


Figure 4.2. Stack Pointer Network for dependency parsing [9]

into the next encoder's input. Finally, the last encoder's output is used for the decoder's prediction.

The transformer model is a baseline for other recent deep learning models such as BERT [55] and it is generally used for machine translation task. There are studies that use BERT for dependency parsing, or there are some models that use the encoder stack and the self-attention. However to our knowledge, there is not any work that directly uses transformer model for dependency parsing as a seq2seq problem. Moreover, there is not any work that uses Transformer model in Turkish dependency parsing. This explains our reason to choose the transformer model in our work.

4.3. Baseline Stack Pointer Model

The Stack-Pointer ($STACKPTR$) model is proposed by Ma et al. (2018) with the paper "Stack-Pointer Networks for Dependency Parsing". The main idea is to combine Pointer Networks [1] with an internal stack in order to predict dependencies in a sentence.

The model contains an encoder and a decoder like in a standard sequence-to-sequence model. The architecture is similar to a Pointer Network [1]. However, Stack Pointer networks additionally contain an internal stack and some high-order information about words in a sentence. Thus, after encoding each sentence, the model processes the word at the top of the internal stack using related high-order information about the word. Then the decoder decodes the encoded information and produces a prediction similar to a Pointer Network.

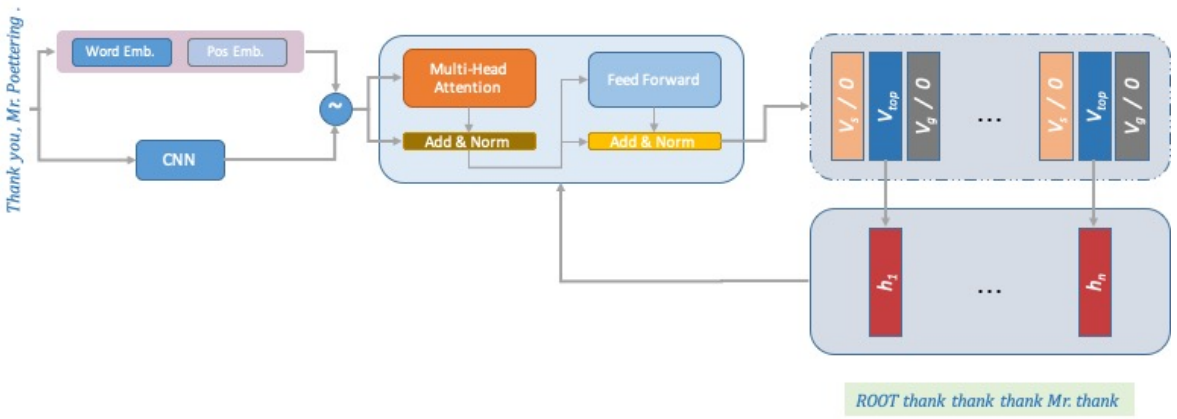


Figure 4.3. Overview of Self Attended Stack Pointer Model

The encoder in $STACKPTR$ contains a BiLSTM-CNN architecture. BiLSTM encoder encodes each word and generates a *hidden state vector* which is called context vector of that word. CNN is used for character-level encoding. Thus, the output of the CNN is a vector that represents character-level encoding of a word. In addition to the character-level word encoding, word-level word embeddings and PoS embeddings are used as input. Each of these vectors are concatenated and the final vector is the input of the BiLSTM encoder.

Once words are encoded by the encoder, the top word in the stack is popped with its high-order information. High-order information is about the word's grandparent and siblings. The hidden state vectors of the current word, the grandparent word, and the sibling words are summed to have the final hidden state vector of the top word.

Eventually, the final vectors are fed into the LSTM decoder. After decoding the sequence, BiAffine classifier [5] is used to predict the heads of the words in the sentence. Cross-Entropy Loss is used for the loss function.

4.4. The Proposed Model: Self Attended Stack Pointer Network Model

Our proposed model, the Self Attended Stack Pointer Network Model, contains an encoder and a decoder stack. In this model, the encoder is adopted from the Transformer Network's Multi-Head Attention encoder and the decoder is adopted from the Stack Pointer's LSTM decoder. The overall architecture is given in Figure 4.3..

Input of the model is a concatenation of word embeddings, PoS embeddings and character-level word embeddings. Character-level word embeddings are obtained from a CNN which encodes each word's character-level information. After the concatenation, we apply Positional Encoding (PE) (see Section 4.4.1.) to this vector, in order to inject positional information in the encoder. Self attended encoder has no information about the positions of words in an input sentence. Hence, we encode the input sentence without losing positional information.

Once the inputs are obtained, the model encodes each input vector in the encoder stack through a Multi-Head Attention, a Feed Forward network and two Layer Normalization steps. Each encoder stack can contain more than one layer of these components. The input of the first encoder layer in the encoder stack is the concatenated vector. Inputs of other encoders are the outputs of the previous encoder layers. Output of each encoder is called as "hidden state vector"s of each word.

Once words are processed within the encoder stack, the model generates the internal stack, which is used to find dependencies between words. We use high-order information of the word at the top of the internal stack. Input of the decoder LSTM is a vector-wised sum of the word's, its sibling's and its grandparent's hidden state vectors. The concatenation is other way to do that, but instead of increasing the complexity by increasing the vector's dimension; we rather use the element-wise summation.

Finally, we decode the highly-ordered hidden state vectors that are generated in the last step and find the possible head-dependent relations using the output of the LSTM decoder and the BiAffine classifier [5]. We predict head-dependent relations by applying softmax on the outputs of the classifier.

The details of the model will be described below.

4.4.1. Positional Encoding

Once the word embeddings are created, we apply Positional Encoding on the embeddings. During the processing of the embeddings in the encoder component of the transformer network, there is no information about the position of words in a sentence. As mentioned before, in the transformer, all words are related with all words. Thus, understanding each word's position is a valuable thing for the model.

So, the authors of the baseline model in paper Vaswani et al. (2017) [8] come up with a simple but effective method, that is positional encoding. It is an injection method applied to embeddings to expose some relative or absolute positions of words in a sentence. This encoding is a basic mathematical expression, actually. The cos function is used for the odd indices and the sin function is used for even indices. The injection of the information of position is performed with the sinus waves. The sin function for even indices is defined as follows:

$$PE(x, 2i) = \sin\left(\frac{x}{10000^{2i/d_{model}}}\right) \tag{11}$$

where d_{model} is the dimension of the word embeddings, $i \in [0, d_{model}/2)$, and x is the position of each word where $x \in [0, n]$ in the input sequence $s = (w_0, w_1 \dots w_n)$.

The cos function is applied for odd indices as given below:

$$PE(x, 2i + 1) = \cos\left(\frac{x}{10000^{2i/d_{model}}}\right) \tag{12}$$

After calculating these values for each value in the embedding, these two values are summed up together. So, the dimension d_{model} does not change after this process. Concatenation is also possible theoretically. However, in the input and output embeddings, the position information is stored in the first few indices in the embedding. Thus, when the d_{model} is high enough, there is no need to concatenate; the summation also meets the requirements.

After adding positional encoding to the embeddings, the encoder stack is initialized. Our model uses 6 layers for both encoder and decoder stacks. However, we experimented with also different number of layers.

Before describing the details of encoder and decoder stacks, first we will explain the self attention mechanism.

4.4.2. Self Attention

Instead of computing the whole sentence's representation, self-attention focuses on different positions and their dependencies in this sentence. Hence, all words are encoded using all words in the sentence. So it learns better relations between words. This all-to-all encoding is performed by using Query, Key and Value matrices.

The self-attention mechanism operates as follows: For a sentence s which contains three words, w_1, w_2 and w_3, in order to find a relation between the words $w1$ and $w2$, w_1 $queries$ w_2 to predict a value that tells the relation between the two words. And the w_2 provides this information in the form of its own key.

When the sentence's embedding ($E_s = [E_{w_1} E_{w_2} E[w_3]]$) is given, first of all, three different but same-sized vectors (Query: Q_{w_i}, Key: K_{w_i} and Value: V_{w_i}) are created for each word in this sentence. Then, scores are calculated with dot product of the query and key values of each word. For example, the dot product of Q_{w_1} with each of K_{w_1}, K_{w_2} and K_{w_3}, creates the main part of the score. After that, the score is divided by the square root of the d_model, the dimension of embeddings, and finally, softmax function is applied to scores in order to make all of them positive and to have a proper probability distribution that sums to 1.

Then, the final scores are multiplied with the value vector of each word. For example, scores that are calculated using Q_{w_1} and K_{w_i} are multiplied with V_{w_i} and it creates V'_{w_i}. This operation reduces the value of irrelevant words. So it makes the relevant words more important. After that operation is applied to each score that is calculated in the previous step, we obtain the weighted values V'_{w_i} for these scores. The summation of these weighted values of each word creates the final embedding for this word. So, new embedding of w_1 is obtained as follows: $E_{w_1} = V'_{w_1} + V'_{w_2} + V'_{w_3}$. As we can see, new embedding of the word contains relevant information about all other words in the sentence.

Self-attention is a base for other attentions in the model such as Multi-Head Attention, Masked Multi-Head Attention and Encoder-Decoder Multi-Head Attention. In the next subsection, we will describe the details of these attentions.

4.4.3. Multi-Head Attention

There are multiple sets of queries, keys and values that are learned in the model. Self-attention is calculated for each of these sets and a new embedding is created. The new embeddings for each set are concatenated and multiplied with Z matrix. Z matrix is trained jointly and multiplied with the concatenated weight matrix in order to reduce the embeddings into a single final embedding for each set. So, in other words, the final embedding is learned from different contexts at the same time.

This operation makes multi-head attention different from the self-attention mechanism. It is multi-head because it learns from the "head" of each set. And the "head" of each set is calculated by using self-attention.

Masked Multi-Head Attention A word in a sentence can depend on the ones before them and the ones after them. Encoding phase of the model actually practices this definition, so all words in a sentence are part of the attention in the encoder stack. However, in decoder stack, we only know the word that we predict; so only the words before the current word are already known at that time step. Because of that, a mask is used at the time of decoding.

The attention in the encoder receives the relations between all words, while the attention in the decoder receives the relations between the first word and the words until the current word. The output embeddings (input embedding of the decoder stack) are shifted one right to predict the next word in a target sequence. Thus, a masked self attention is applied in the decoder.

The model is able to learn without applying any masking. However, in that case it tries to decode something that it does not know anything. So it brings ambiguity to the model.

The output of the Encoder stack contains the input embedding (input of the model), positional information and contextual information. In other words, the output of the encoder stack is an improved version of the input embedding. When we feed this embedding to the self-attention with the embedding obtained from the masked multi-head attention, we can predict the next word in a target sequence with the following features:

- Standard Input Embedding
- Positional Information
- Contextual information
- Output Embedding (until the current word)

4.4.4. Layer Normalization

Layer Normalization [102] is used to normalize the input across various features. It is more consistent compared to batch normalization because, in batch normalization, features are

normalized across a batch. Thus, it imposes a lower bound on the batch size. However, in layer normalization, the computed values are for each feature and it is independent from the other examples.

In the model, we use layer normalization after each attention and feed-forward neural network because it normalizes the weights and retain some form of information from the previous layers.

4.4.5. Feed-Forward Neural Network

In the model, there is one feed-forward neural network in encoder and one in decoder. It is basically a neural network with two Linear layers and ReLU activation function. It is placed at the end of the encoder and decoder because with this feed-forward neural network, we can train the embeddings with a latent space of words.

4.4.6. Encoder

As we mentioned above, an Encoder stack contains one or more encoders which contains a Masked Multi-Head attention, a Feed-Forward Neural Network and Layer Normalization after each of them. If there is one layer in encoder stack, then the output of this encoder directly goes to Encoder-Decoder attention of each layer in Decoder stack. If there are many layers in the encoder stack, then each encoder's output is an input for the next layer except the last one. The last one always goes to Encoder-Decoder attention of each layer in Decoder stack.

Using many layers is a rational choice, because if the encoder improves the input embeddings; output of the second layer is the improved version of output of the first layer and so on. However, while the number of layers increases, the feature space also increases and it requires a larger dataset. That will bring some computational issues in the model.

4.4.7. Higher-order Information

Before decoding, the model operates hidden state vectors, which are the outputs of the encoder. During encoding, hidden state vectors are created for all words in a sentence. But the

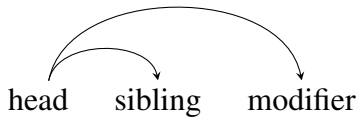


Figure 4.4. Sibling structure

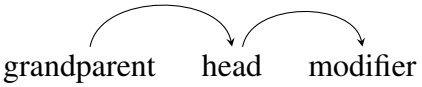


Figure 4.5. Grandchild structure

most important word at a time stamp is the word at the top of stack. Thus, creating a vector with some higher-order information about the word at the top makes the hidden state vector more efficient.

For that purpose, Ma et al. (2018)[9] are inspired from the model of McDonald and Pereira (2006)[103] and propose to use some higher-order information such as siblings and grandchilds.

In the sibling structure given in Figure 4.4., a head word is the head of two different modifiers. In other words, if the word is sharing the same head with another modifier, then the word is a sibling of this modifier.

In the grandchild structure given in Figure 4.5., there is a head-to-tail connection. In other words, the grandparent of a word is the head of this word's head.

Thus, the input of the decoder is the element-wise sum of the encoder hidden state s_h, sibling hidden state s_s and grandchild hidden state vectors s_g:

$$\beta_i = s_h + s_s + s_g \tag{13}$$

where β_i is the input vector of decoder.

4.4.8. Decoder

A decoder contains an LSTM structure and a BiAffine classifier[5] to process all contextual and higher-order information about the word at the top of the stack.

Normally, in a pointer network, at each time step t, the decoder receives the input and outputs decoder hidden state h_t, as follows:

$$e_i^t = score(h_t, s_i) \tag{14}$$

where e^t is the output of the scoring function, s_i is the encoder hidden state and h_t is the decoder hidden state. In our model, scoring function is adopted from Deep BiAffine attention mechanism [5]:

$$e_i^t = h_t^T W s_i + U^t h_t + V^t s_i + b \tag{15}$$

where W is the weight matrix, U and V are weight vectors and b is the bias. Before computing the scores, an MLP is applied to the output of the decoder as discussed in Dozat and Manning (2016) [5]. After calculating the score for the potential outputs, the final prediction is made by using softmax:

$$a^t = softmax(e^t) \tag{16}$$

where a^t is the output vector and e^t is the output of the scoring function.

We use cross-entropy loss similar to $STACKPTR$. The probability of a head word given a word in a sentence, $P_\theta(y|x)$ is computed as follows:

$$\begin{aligned} P_\theta(y|x) &= \prod_{i=1}^{k} P_\theta(p_i|p_{<i}, x) \\ &= \prod_{i=1}^{k} \prod_{j=1}^{l_i} P_\theta(c_{i,j}|c_{i,<j}, p_{<i}, x) \end{aligned} \tag{17}$$

where θ denotes the model parameters, x denotes the sentence, y denotes the parse tree, $p_{<i}$ denotes the preceding paths that have already been generated, $c_{i,j}$ represents the j^{th} word in p_i and $c_{i,<j}$ denotes all the following words on the path p_i. A path is defined as all words from a root till the leaf node in a parse tree.

The model learns the arcs and labels in the dependency tree simultaneously.

5. EXPERIMENTS AND RESULTS

In this chapter, we give information about the datasets, pre-trained embeddings and evaluation metrics which are used in this thesis and we present results obtained from our proposed model. Finally, we compare the results with that of other related work on dependency parsing and discuss the performance of the model depending on various features such as the length of sentences.

5.1. Datasets

We ran experiments on both Turkish and English. Following subsections contain information about the dependency datasets and the word embeddings used in the experiments.

5.1.1. Penn Treebank

Penn Treebank (PTB) [79] is an English dependency treebank which is obtained from various sources such as Wall Street Journal (WSJ), the Air Traffic Information System, etc. The dependency treebank contains non-projective and projective trees. However, the majority is composed of projective sentences.

5.1.2. Universal Dependencies

Universal Dependencies (UD) is a framework created by an open community which contains 150 treebanks for around 90 languages. We used the IMST dataset [63] in Universal Dependencies for Turkish.

5.1.3. GloVe Embeddings

Global Vectors for Word Representation (GloVe) [104] is an unsupervised method for extracting vector representations of words. We use 200-dimensional GloVe vectors, which are pre-trained on Wikipedia for both Turkish and English.

5.1.4. Polyglot Embeddings

Polyglot [105] contains distributed word embeddings for more than one hundred languages. Raw data is mainly obtained from Wikipedia data. The Polyglot embeddings give the best results (compared with other embeddings) on languages such as English and Swedish for sequence tagging task.

5.2. Evaluation Metrics

5.2.1. Unlabeled Attachment Score (UAS)

UAS is a metric that is used to calculate the accuracy of predicting words' heads. In other words, it is the ratio of the number of correctly predicted heads to the total number of words in the dataset:

$$\frac{UAS = \#ofcorrectheads}{\#ofwords} \tag{18}$$

5.2.2. Labeled Attachment Score (LAS)

LAS is another metric for dependency parsing that measures the correctness of both heads and labels. In other words, it is the ratio of correctly predicted heads **and** labels to the total number of words in the dataset:

$$\frac{LAS = \#ofcorrecthead, labelpair}{\#ofwords} \tag{19}$$

5.2.3. Root Accuracy (RA)

RA is used to measure the prediction accuracy of the $ROOT$ word. In parsing, each sentence has one $ROOT$ word and correct prediction of the $ROOT$ word plays a crucial role in the prediction of other words' dependencies in a sentence.

Model	UAS	LAS
Our Model w/ Glove[104]	74.43	64.26
Our Model w/ Polyglot[105]	76.81	67.95
Nguyen and Verspoor (2018)[53]	70.53	62.55
Kondratyuk and Straka (2019)[54]	74.56	67.44
McDonald et al. (2006)[42]	74.70	63.20
Dozat and Manning (2016)[5]	77.46	68.02
Ballesteros et al. (2015)[60]	79.30	69.28
Ma et al. (2018)[9]	79.56	68.93

Table 5.1. Results for Turkish IMST Dataset [63]

5.3. Experiments

We ran experiments for both English and Turkish using Penn Treebank dataset [79] and IMST dataset [63] respectively.

5.3.1. Turkish

For Turkish, we used the Turkish IMST dataset [63] from CONLL 2018 Shared Task [52] along with Polyglot [105] and Glove [104] analogously to the English experiments. Using Polyglot embeddings gives better results in Turkish, too. Our model gives an UAS score of 74.43% and LAS score of 64.26% with Glove embeddings, whereas an UAS score 76.81% and LAS score of 67.95% with Polyglot embeddings. The most important reason of Polyglot's impact on the performance of dependency parsing comes from the number of unique words that it contains. For Turkish, while the Glove was trained on 13411 words, Polyglot was trained on 15104 words.

Our model comes the 4^{th} out of 8 dependency parsers.

The main aim in this thesis behind utilizing Transformer Networks is to resolve the long-term dependencies problem in parsing. We analyzed the accuracy of our model in both short and longer sentences to see the impact of the Transformer Networks in model.

Our model performs better than our baseline model, the $STACKPTR$, in longer sentences.

An example sentence from IMST dataset [63] contains 38 words:

Word	Gold	Prediction - StackPTR	Prediction - Self-Attended StackPTR
Bu	iş / det	**iş / det**	**iş / det**
iş	olmayacaktı / nsubj	**olmayacaktı** / obj	**olmayacaktı / nsubj**
böyle	olmayacaktı / advmod	**olmayacaktı / advmod**	**olmayacaktı / advmod**
olmayacaktı	ROOT	**ROOT**	**ROOT**
,	giriyorduk / punct	**giriyorduk / punct**	**giriyorduk / punct**
çünkü	giriyorduk / cc	**giriyorduk / cc**	**giriyorduk / cc**
çok	sık / advmod	**sık / advmod**	**sık / advmod**
sık	giriyorduk / amod	**giriyorduk / amod**	**giriyorduk / amod**
belimize*	suya / nmod	giriyorduk / obl	giriyorduk / obl
dek	belimize / case	**belimize / case**	**belimize / case**
suya*	giriyorduk / obl	**giriyorduk** / amod	**giriyorduk / obl**
giriyorduk*	olmayacaktı / conj	**olmayacaktı / conj**	**olmayacaktı / conj**
ve	. / cc	sevgililerin / **cc**	**. / cc**
kayalara	serpiştirilmiş / obl	**serpiştirilmiş / obl**	**serpiştirilmiş / obl**
serpiştirilmiş	dağılmış / acl	**dağılmış / acl**	**dağılmış / acl**
gibi	serpiştirilmiş / case	**serpiştirilmiş / case**	**serpiştirilmiş / case**
birbirlerinden	uzaklıklara / nmod	dağılmış / obl	**uzaklıklara / nmod**
belli	uzaklıklara / amod	**uzaklıklara / amod**	**uzaklıklara** / nmod
uzaklıklara	dağılmış / obl	**dağılmış / obl**	**dağılmış / obl**
dağılmış	sevgililerin / nmod	**sevgililerin** / acl	**sevgililerin** / acl
sevgililerin	rahatsız / compound	olmayacaktı / conj	**rahatsız / compound**
(	yaz / punct	başlamıştık / **punct**	**yaz / punct**
yaz*	sevgililerin / appos	aşkları / amod	**sevgililerin / appos**
aşkları	yaz / compound	bunlar / **compound**	bunlar / **compound**
bunlar	yaz / nsubj	başlamıştık / **nsubj**	**yaz / nsubj**
,	düşünmeye / punct	çakıyoruz / **punct**	**düşünmeye / punct**
o	kadarını / det	**kadarını / det**	**kadarını / det**
kadarını*	çakıyoruz / obj	**çakıyoruz / obj**	**çakıyoruz / obj**
çakıyoruz*	yaz / conj	olmayacaktı / **conj**	**yaz / conj**
)	yaz / punct	çakıyoruz / **punct**	çakıyoruz / **punct**
,	çakıyoruz / punct	başlamıştık / **punct**	başlamıştık / **punct**
bu	durumdan / det	**durumdan / det**	**durumdan / det**
durumdan	rahatsız / obl	olacaklarını / **obl**	**rahatsız / obl**
rahatsız	düşünmeye / obj	olacaklarını / amod	olacaklarını / amod
olacaklarını	rahatsız / compound:lvc	düşünmeye / obj	**rahatsız** / obj
düşünmeye	başlamıştık / nmod	**başlamıştık / nmod**	**başlamıştık / nmod**
başlamıştık*	olmayacaktı / conj	çakıyoruz / **conj**	**olmayacaktı / conj**
.	başlamıştık / punct	**başlamıştık / punct**	**başlamıştık / punct**

Table 5.2. Prediction of the heads by $STACKPTR$ [9] and our model Self-Attended $STACKPTR$ for a sample long sentence in Turkish IMST Dataset [63]

> *Bu iş böyle olmayacaktı, çünkü çok sık belimize dek suya giriyorduk ve kayalara serpiştirilmiş gibi birbirlerinden belli uzaklıklara dağılmış sevgililerin (yaz aşkları bunlar, o kadarını çakıyoruz), bu durumdan rahatsız olacaklarını düşünmeye başlamıştık.*

This sentence contains more than one verbal. Moreover, as we can see in "(yaz aşkları bunlar, o kadarını çakıyoruz)", it contains other sentences which describes the main sentence. Thus, parsing this sentence is a bit tricky. Table 5.2. contains the gold heads of words in that sentence with the prediction of the Ma et al. (2018) ($STACKPTR$) which is the best model in Table 5.1. and our proposed model with Transformer Networks, which is named Self-Attended StackPTR.

Number of words in sentence	UAS - StackPTR	UAS - Self-Attended StackPTR
less than 10 words	93.23	86.47
between 10 and 20 words	88.96	81.63
more than 20 words	56.49	62.33

Table 5.3. Success rate by word counts for IMST Dataset[63]

In Table 5.2., the bold lines at the rightmost column are the head/label pairs which are predicted correctly by our model. As we can see, the wrong predictions are generally between the first comma and last comma in the sentence, because those words between them are inner sentences and verbals. Also, around the phrase "(yaz aşkları bunlar, o kadarını çakıyoruz)", the difference in the predictions is even more.

For example, for the word "*başlamıştık*", the $STACKPTR$ model predicts the head as "*çakıyoruz*" which is the final word between the parentheses. However, correct prediction is the fourth word, "*olmayacaktı*", which is the last word before the first comma. In other words, while the correct prediction is the first verbal, $STACKPTR$ model predicts the last verbal. However, our model is able to correctly detect the dependency between these two words.

Table 5.3. gives the results for different lengths of sentences to show the impact of using Transformer Networks in longterm depedencies. We compare our model with the original $STACKPTR$ [9] model, which is based on LSTMs rather than Transformer Networks. As the results show, our model performs better for sentences with more than 20 words compared to the standard $STACKPTR$ model. This is the most important benefit of Self-Attention mechanism.

For less than 20 words, our model's accuracy is lower compared to longer sentences. It shows that our self-attention based model is not able to learn shorter sentences better than the BiLSTM based $STACKPTR$ model. However, we observed that decreasing the number of layers in our encoder stack gives a higher accuracy for shorter sentences. However, it decreases the overall accuracy for the entire dataset. It is possible that instead of decreasing the number of layers, if we increase the size of the dataset, our model may outperform the $STACKPTR$ model.

Additionally, we analyzed the Root Accuracy (RA) for Turkish. The Root Accuracy (RA) of our model is 82.47%.

Dataset/Version	w/ Punctuation	w/o Punctuation
PTB	94.23 (92.67)	93.47 (91.94)
IMST	76.81 (67.95)	71.96 (62.41)

Table 5.4. Accuracy (UAS (LAS)) by punctuations for IMST[63] and PTB[79] Datasets

Score/Projectivity	Projective	Non-Projective
UAS	77.69	60.17
LAS	68.47	58.21

Table 5.5. Accuracy by projectivity for IMST[63] Dataset

We also analyzed the impact of using punctuation in the datasets during training. Analysis of Spitkovsky et al. [106] shows that the usage of lexicalized and punctuated sentences give better results in dependency parsing. So, we ran our model with both punctuated and not-punctuated versions of both datasets in Turkish and English.

Table 5.4. shows that punctuation affects the learning of the model for both languages and the results are comparably higher when the punctuation is also used in the datasets. The impact of using punctuation is even more for Turkish language and both UAS and LAS are around %5 higher compared to using datasets without punctuation.

Moreover, one of our other experiments is about projectivity of the predictions. In IMST dataset [63], there are 58085 words in total that consist of 55043 projective and 3042 non-projective words; in other words, approximately 5% of the dataset is non-projective. This ratio is almost the same for both train, dev and test datasets.

Our model, which contains BiAffine classifier, produces mostly projective trees. Thus, the performance for non-projective trees is slightly worse; because it tries to find a projective tree for these sentences and it fails. Table 5.5. shows our model's performance on projective and non-projective sentences for IMST [63] dataset.

As explained in the Section 4.4., we utilize word embeddings, PoS tag embeddings and char embeddings obtained from CNN in our model. Table 5.6. shows the impact of the embeddings on the accuracy of the model. Character-level encoding plays a crucial role in our model because it helps to mitigate the OOV problem during training. We obtained the highest scores when Polyglot word embeddings, PoS tag embeddings and character-based word embeddings are incorporated in training.

Input Embeddings	UAS
Glove	63.24
Polyglot	65.76
Polyglot + PoS	70.48
Polyglot + CNN	73.81
Polyglot + PoS + CNN	76.81

Table 5.6. The impact of using word embeddings (Glove or Polyglot), PoS tag embeddings and character-based word embeddings for the Turkish IMST Dataset [63]

Model	UAS	LAS
Our Model w/ Glove[104]	93.43	91.98
Our Model w/ Polyglot[105]	94.23	92.67
Ballesteros et al. (2015)[60]	91.63	89.44
Chen and Manning (2014)[7]	91.8	89.6
Kiperwasser and Goldberg (2016)[64]	93.1	91.0
Ballesteros et al. (2016)[93]	93.56	91.42
Weiss et al. (2015)[87]	94.26	92.41
Andor et al. (2016)[91]	94.61	92.79
Ma and Hovy (2017)[108]	94.88	92.98
Dozat and Manning (2016)[5]	95.74	94.08
Ma et al. (2018)[9]	95.87	94.19

Table 5.7. Results for English PTB Dataset [79]

In addition to these emeddings, we can consider using morpheme embeddings (i.e. Morph2Vec[107]) and/or inflection groups (IGs)[2], which remain as future work.

5.3.2. English

For the experiments in English, we used Penn Treebank dataset [79] along with Polyglot [105] or Glove [104] embeddings. Our model performs better with Polyglot embeddings. While Glove gives 93.43% UAS and 91.98%LAS, Polyglot gives 94.23% UAS and 92.67% LAS.

Table5.7. contains a comparison of our model and other related work in dependency parsing.

Number of words in sentence	UAS - StackPTR	UAS - Self-Attended StackPTR
less than 10 words	97.42	95.92
between 10 and 20 words	95.38	92.85
more than 20 words	92.61	93.25

Table 5.8. Success rate by word counts for PTB Dataset[79]

Layer	UAS
1	86.24
2	88.56
4	92.40
6	**94.23**
8	93.13

Table 5.9. Accuracy of encoder layers for PTB Dataset[79]

We can see from the table 5.7. that our model comes the 5th out of 10 other dependency parsers for English language.

Rate of success by sentence length can be found in Table 5.8..

Our results show that our model performs better for the longer sentences in English, too.

5.4. Implementation Details

In our experiments, we use similar configurations with the baseline models: $STACKPTR$ model by Ma et al. (2018) [9] and Self-Attention mechanism by Vaswani et al. (2017) [8]. Differently from the baseline models, for the self-attended encoder stack; we use 6 layers because this configuration performs better with the Polyglot embeddings as can be seen in Table 5.9. and Table 5.10. for both English and Turkish respectively.

We implemented the model using PyTorch [109] framework in addition to NumPy[110] and Gensim[111] libraries in Python 3. Our experiments were ran on Google Cloud server with 4 vCPUs and 15 GB memory. Details of the implementation can be found here: `https://github.com/salihtuc/self-attn-stackptr`.

Layer	UAS
1	69.89
2	71.48
4	74.51
6	**76.81**
8	75.32

Table 5.10. Accuracy of encoder layers for Turkish IMST Dataset [63]

5.5. Conclusion

Our experiments show that using Self-Attention mechanism increases parsing accuracy especially in longer sentences in Turkish. However, our parser requires more data to learn better for also shorter sentences.

The results also show that using character level word embeddings along with word embeddings and PoS tag embeddings gives the highest accuracy for our model.

We obtained the highest scores when we include 6 layers in our encoder stack by using Polyglot embeddings. Our results also show that including punctuation in the dataset improves the accuracy substantially.

6. CONCLUSION

6.1. Conclusion

In this work, we proposed a novel neural encoder-decoder architecture, which is called Self-Attended Pointer Networks. We utilized Self-Attention mechanism instead of a recurrent structure such as a LSTM or RNN for encoding the sequences and used Stack Pointer Networks for decoding the sequence to obtain the dependencies . Hence, our model consists of two components: Self-Attention encoder with a Transformer Network and LSTM-based decoder with BiAffine classifier.

Our experiments show that our proposed model performs better in longer sentences (which has more than 20 words) compared to a LSTM-based encoder-decoder structure. This shows that self-attention mechanism handles long term dependencies efficiently. However, the overall results for our model is not the best because the self-attention mechanism needs more data during training. Moreover, our method is the first Self-Attention based model used in Turkish dependency parsing.

6.2. Future Research Directions

Our results show that using Transformer Networks leads to a better performance especially for long term dependencies. However, our model gives an average accuracy for the entire dataset for both Turkish and English. In consideration of the morphological structure of Turkish and the heavy usage of suffixes and inflection groups in Turkish, we plan to incorporate morpheme embeddings in the model. Moreover, we plan to investigate to define dependencies between inflectional groups or morphemes rather than between word tokens.

As we mentioned in the previous sections, the self-attention mechanism needs more data compared to recurrent neural networks because it obtains information about all words from all words in a dataset. If we increase the size of the dataset or find additional ways to increase contextual relations between words in the dataset, we believe that our results will be better.

REFERENCES

[1] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks, **2015**.

[2] Gülşen Eryiğit, Joakim Nivre, and Kemal Oflazer. Dependency parsing of turkish. *Computational Linguistics*, 34(3):357–389, **2008**.

[3] Kemal Oflazer. Turkish and its challenges for language processing. *Lang. Resour. Eval.*, 48(4):639–653, **2014**. ISSN 1574-020X. doi:10.1007/s10579-014-9267-2.

[4] Zuchao Li, Jiaxun Cai, Shexia He, and Hai Zhao. Seq2seq dependency parsing. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3203–3214. Association for Computational Linguistics, Santa Fe, New Mexico, USA, **2018**.

[5] Timothy Dozat and Christopher D. Manning. Deep biaffine attention for neural dependency parsing, **2016**.

[6] Junru Zhou and Hai Zhao. Head-driven phrase structure grammar parsing on Penn treebank. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2396–2408. Association for Computational Linguistics, Florence, Italy, **2019**. doi:10.18653/v1/P19-1230.

[7] Danqi Chen and Christopher Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750. Association for Computational Linguistics, Doha, Qatar, **2014**. doi:10.3115/v1/D14-1082.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, **2017**.

[9] Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*

(Volume 1: Long Papers), pages 1403–1414. Association for Computational Linguistics, Melbourne, Australia, **2018**. doi:10.18653/v1/P18-1130.

[10] Xavier Carreras and Michael Collins. Non-projective parsing for statistical machine translation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 200–209. Association for Computational Linguistics, Singapore, **2009**.

[11] Huadong Chen, Shujian Huang, David Chiang, and Jiajun Chen. Improved neural machine translation with a syntax-aware encoder and decoder. pages 1936–1945. **2017**. doi:10.18653/v1/P17-1177.

[12] Katrin Fundel-Clemens, Robert Küffner, and Ralf Zimmer. Relex - relation extraction using dependency parse trees. *Bioinformatics (Oxford, England)*, 23:365–71, **2007**. doi:10.1093/bioinformatics/btl616.

[13] Yuhao Zhang, Peng Qi, and Christopher D. Manning. Graph convolution over pruned dependency trees improves relation extraction. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2205–2215. Association for Computational Linguistics, Brussels, Belgium, **2018**. doi:10.18653/v1/D18-1244.

[14] Zhanming Jie, Aldrian Obaja Muis, and Wei Lu. Efficient dependency-guided named entity recognition. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 3457–3465. AAAI Press, **2017**.

[15] Jenny Rose Finkel and Christopher D. Manning. Joint parsing and named entity recognition. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 326–334. Association for Computational Linguistics, Boulder, Colorado, **2009**.

[16] Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D. Manning. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 344–354. Association for Computational Linguistics, Beijing, China, **2015**. doi:10.3115/v1/P15-1034.

[17] Nanyun Peng, Hoifung Poon, Chris Quirk, Kristina Toutanova, and Wen-tau Yih. Cross-sentence n-ary relation extraction with graph lstms. *Transactions of the Association for Computational Linguistics*, 5, **2017**. doi:10.1162/tacl_a_00049.

[18] Marie-Catherine de Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D. Manning. Universal Stanford dependencies: A cross-linguistic typology. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 4585–4592. European Language Resources Association (ELRA), Reykjavik, Iceland, **2014**.

[19] Julia Hockenmaier and Mark Steedman. Ccgbank: A corpus of ccg derivations and dependency structures extracted from the penn treebank. *Computational Linguistics*, 33(3):355–396, **2007**. ISSN 0891-2017. doi:10.1162/coli.2007.33.3.355.

[20] David G Hays. Dependency theory: A formalism and some observations. *Language*, 40(4):511–525, **1964**.

[21] L Tesnière. Éléments de syntaxe structurale. **1959**.

[22] W. Keith Percival. Reflections on the history of dependency notions in linguistics. *Historiographia Linguistica*, 17(1-2):29–47, **1990**. ISSN 0302-5160. doi:https://doi.org/10.1075/hl.17.1-2.05per.

[23] Jack Edmonds. Optimum branchings.

[24] Jason M. Eisner. Three new probabilistic models for dependency parsing: An exploration. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*. **1996**.

[25] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, **2014**.

[26] Rohit Prabhavalkar, Kanishka Rao, Tara Sainath, Bo Li, Leif Johnson, and Navdeep Jaitly. A comparison of sequence-to-sequence models for speech recognition. **2017**.

[27] Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. Unified language model pre-training for natural language understanding and generation, **2019**.

[28] Augustin Cauchy. Méthode générale pour la résolution des systemes d'équations simultanées.

[29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, **2014**.

[30] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12(null):2121–2159, **2011**. ISSN 1532-4435.

[31] Gulsen Eryigit and Kemal Oflazer. Statistical dependency parsing of turkish. **2006**.

[32] Joakim Nivre, Johan Hall, Jens Nilsson, Gülşen Eryiğit, and Svetoslav Marinov. Labeled pseudo-projective dependency parsing with support vector machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 221–225. Association for Computational Linguistics, New York City, **2006**.

[33] Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülşen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135, **2007**.

[34] Gülşen Eryiğit. The impact of automatic morphological analysis & disambiguation on dependency parsing of Turkish. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 1960–1965. European Language Resources Association (ELRA), Istanbul, Turkey, **2012**.

[35] Ruket Cakıcı and Jason Baldridge. Projective and non-projective turkish parsing. **2006**.

[36] Michael Collins. Three generative, lexicalised models for statistical parsing. In *35th Annual Meeting of the Association for Computational Linguistics and*

8th Conference of the European Chapter of the Association for Computational Linguistics, pages 16–23. Association for Computational Linguistics, Madrid, Spain, **1997**. doi:10.3115/976909.979620.

[37] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan 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*, pages 523–530. Association for Computational Linguistics, Vancouver, British Columbia, Canada, **2005**.

[38] Özlem Çetinoğlu and Jonas Kuhn. Towards joint morphological analysis and dependency parsing of Turkish. In *Proceedings of the Second International Conference on Dependency Linguistics (DepLing 2013)*, pages 23–32. Charles University in Prague, Matfyzpress, Prague, Czech Republic, Prague, Czech Republic, **2013**.

[39] Deniz Yuret. Discovery of linguistic relations using lexical attraction, **1998**.

[40] Daniel Fernández-González and Carlos Gómez-Rodríguez. Non-projective dependency parsing with non-local transitions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 693–700. Association for Computational Linguistics, New Orleans, Louisiana, **2018**. doi:10.18653/v1/N18-2109.

[41] Michael A Covington. A fundamental algorithm for dependency parsing. Citeseer, **2001**.

[42] Ryan McDonald, Kevin Lerman, and Fernando Pereira. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 216–220. Association for Computational Linguistics, New York City, **2006**.

[43] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, **1967**. ISSN 1557-9654. doi:10.1109/TIT.1967.1054010.

[44] Sabine Buchholz and Erwin Marsi. Conll-x shared task on multilingual dependency parsing. In *Proceedings of the tenth conference on computational natural language learning (CoNLL-X)*, pages 149–164. **2006**.

[45] Ivan Titov and James Henderson. Fast and robust multilingual dependency parsing with a generative latent variable model. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 947–951. Association for Computational Linguistics, Prague, Czech Republic, **2007**.

[46] Ivan Titov and James Henderson. Constituent parsing with incremental sigmoid belief networks. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 632–639. Association for Computational Linguistics, Prague, Czech Republic, **2007**.

[47] Radford M Neal. Connectionist learning of belief networks. *Artificial intelligence*, 56(1):71–113, **1992**.

[48] Ivan Titov and James Henderson. A latent variable model for generative dependency parsing. In *Proceedings of the Tenth International Conference on Parsing Technologies*, pages 144–155. Association for Computational Linguistics, Prague, Czech Republic, **2007**.

[49] Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 915–932. Association for Computational Linguistics, Prague, Czech Republic, **2007**.

[50] Arda Akdemir and Tunga Güngör. Joint learning of named entity recognition and dependency parsing using separate datasets. *Computación y Sistemas*, 23(3), **2019**.

[51] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. **2001**.

[52] Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav 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*, pages 1–21. Association for Computational Linguistics, Brussels, Belgium, **2018**. doi:10.18653/v1/K18-2001.

[53] Dat Quoc Nguyen and Karin Verspoor. An improved neural network model for joint. *Proceedings of the*, **2018**. doi:10.18653/v1/k18-2008.

[54] Dan Kondratyuk and Milan Straka. 75 languages, 1 model: Parsing universal dependencies universally, **2019**.

[55] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, **2018**.

[56] Milan Straka. UDPipe 2.0 prototype at CoNLL 2018 UD shared task. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 197–207. Association for Computational Linguistics, Brussels, Belgium, **2018**. doi:10.18653/v1/K18-2020.

[57] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations, **2018**.

[58] Wanxiang Che, Yijia Liu, Yuxuan Wang, Bo Zheng, and Ting Liu. Towards better ud parsing: Deep contextualized word embeddings, ensemble, and treebank concatenation, **2018**.

[59] Artur Kulmizev, Miryam de Lhoneux, Johannes Gontrum, Elena Fano, and Joakim Nivre. Deep contextualized word embeddings in transition-based and graph-based dependency parsing – a tale of two parsers revisited, **2019**.

[60] Miguel Ballesteros, Chris Dyer, and Noah A. Smith. Improved transition-based parsing by modeling characters instead of words with lstms, **2015**.

[61] Utku Türk, Furkan Atmaca, Şaziye Betül Özateş, Balkız Öztürk Başaran, Tunga Güngör, and Arzucan Özgür. Improving the annotations in the Turkish universal dependency treebank. In *Proceedings of the Third Workshop on Universal*

Dependencies (UDW, SyntaxFest 2019), pages 108–115. Association for Computational Linguistics, Paris, France, **2019**. doi:10.18653/v1/W19-8013.

[62] Bharat Ram Ambati, Siva Reddy, and Adam Kilgarriff. Word sketches for turkish.

[63] Umut Sulubacak, Memduh Gokirmak, Francis Tyers, Çağrı Çöltekin, Joakim Nivre, and Gülşen Eryiğit. Universal dependencies for Turkish. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 3444–3454. The COLING 2016 Organizing Committee, Osaka, Japan, **2016**.

[64] Eliyahu Kiperwasser and Yoav Goldberg. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327, **2016**. doi:10.1162/tacl_a_00101.

[65] Eliyahu Kiperwasser and Yoav Goldberg. Easy-first dependency parsing with hierarchical tree LSTMs. *Transactions of the Association for Computational Linguistics*, 4:445–461, **2016**. doi:10.1162/tacl_a_00110.

[66] Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard 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*, pages 1923–1933. Association for Computational Linguistics, Copenhagen, Denmark, **2017**. doi:10.18653/v1/D17-1206.

[67] Hao Cheng, Hao Fang, Xiaodong He, Jianfeng Gao, and Li Deng. Bi-directional attention with agreement for dependency parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2204–2214. Association for Computational Linguistics, Austin, Texas, **2016**. doi:10.18653/v1/D16-1238.

[68] Ying Li, Zhenghua Li, Min Zhang, Rui Wang, Sheng Li, and Luo Si. Self-attentive biaffine dependency parsing. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 5067–5073. AAAI Press, **2019**.

[69] Carl Pollard and Ivan A Sag. *Head-driven phrase structure grammar*. University of Chicago Press, **1994**.

[70] Nikita Kitaev and Dan Klein. Constituency parsing with a self-attentive encoder. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2676–2686. Association for Computational Linguistics, Melbourne, Australia, **2018**. doi:10.18653/v1/P18-1249.

[71] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding, **2019**.

[72] Do Kook Choe and Eugene Charniak. Parsing as language modeling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2331–2336. Association for Computational Linguistics, Austin, Texas, **2016**. doi:10.18653/v1/D16-1257.

[73] Tao Ji, Yuanbin Wu, and Man Lan. Graph-based dependency parsing with graph neural networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2475–2485. Association for Computational Linguistics, Florence, Italy, **2019**. doi:10.18653/v1/P19-1237.

[74] John Hewitt and Christopher 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)*, pages 4129–4138. Association for Computational Linguistics, Minneapolis, Minnesota, **2019**. doi:10.18653/v1/N19-1419.

[75] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566. Association for Computational Linguistics, Beijing, China, **2015**. doi:10.3115/v1/P15-1150.

[76] Kevin Clark, Minh-Thang Luong, Christopher D. Manning, and Quoc V. Le. Semi-supervised sequence modeling with cross-view training, **2018**.

[77] Peng Qi, Timothy Dozat, Yuhao Zhang, and Christopher D. Manning. Universal dependency parsing from scratch, **2019**.

[78] Phong Le and Willem Zuidema. Unsupervised dependency parsing: Let's use supervised parsers. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 651–661. Association for Computational Linguistics, Denver, Colorado, **2015**. doi:10.3115/v1/N15-1067.

[79] Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. **1993**.

[80] Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. Breaking out of local optima with count transforms and model recombination: A study in grammar induction. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1983–1995. Association for Computational Linguistics, Seattle, Washington, USA, **2013**.

[81] Dan Klein and Christopher Manning. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 478–485. Barcelona, Spain, **2004**. doi:10.3115/1218955.1219016.

[82] Noah A Smith and Jason Eisner. Guiding unsupervised grammar induction using contrastive estimation.

[83] William P. Headden III, Mark Johnson, and David McClosky. Improving unsupervised dependency parsing with richer contexts and smoothing. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 101–109. Association for Computational Linguistics, Boulder, Colorado, **2009**.

[84] John Cocke. Programming languages and their compilers: Preliminary notes. **1970**.

[85] Daniel H. Younger. Recognition and parsing of context-free languages in time n3. *Information and Control*, 10(2):189 – 208, **1967**. ISSN 0019-9958. doi: https://doi.org/10.1016/S0019-9958(67)80007-X.

[86] Tadao Kasami. An efficient recognition and syntax-analysis algorithm for context-free languages. *Coordinated Science Laboratory Report no. R-257*, **1966**.

[87] David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. Structured training for neural network transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 323–333. Association for Computational Linguistics, Beijing, China, **2015**. doi:10.3115/v1/P15-1032.

[88] Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 334–343. Association for Computational Linguistics, Beijing, China, **2015**. doi:10.3115/v1/P15-1033.

[89] Xinchi Chen, Yaqian Zhou, Chenxi Zhu, Xipeng Qiu, and Xuanjing Huang. Transition-based dependency parsing using two heterogeneous gated recursive neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1879–1889. Association for Computational Linguistics, Lisbon, Portugal, **2015**. doi:10.18653/v1/D15-1215.

[90] Alireza Mohammadshahi and James Henderson. Graph-to-graph transformer for transition-based dependency parsing, **2019**.

[91] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452. Association for Computational Linguistics, Berlin, Germany, **2016**. doi:10.18653/v1/P16-1231.

[92] Daniel Fernández-González and Carlos Gómez-Rodríguez. Left-to-right dependency parsing with pointer networks. In *Proceedings of the 2019 Conference*

of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pages 710–716. Association for Computational Linguistics, Minneapolis, Minnesota, **2019**. doi:10.18653/v1/N19-1076.

[93] Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A. Smith. Training with exploration improves a greedy stack-lstm parser, **2016**.

[94] Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, and Noah A. Smith. Distilling an ensemble of greedy dependency parsers into one mst parser, **2016**.

[95] Peng Qi and Christopher D. Manning. Arc-swift: A novel transition system for dependency parsing, **2017**.

[96] Joakim Nivre and Ryan McDonald. Integrating graph-based and transition-based dependency parsers. In *Proceedings of ACL-08: HLT*, pages 950–958. Association for Computational Linguistics, Columbus, Ohio, **2008**.

[97] Yoav Goldberg and Michael Elhadad. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750. Association for Computational Linguistics, Los Angeles, California, **2010**.

[98] Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. From baby steps to leapfrog: How "less is more" in unsupervised dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 751–759. Association for Computational Linguistics, Los Angeles, California, **2010**.

[99] Ji Ma, Jingbo Zhu, Tong Xiao, and Nan Yang. Easy-first POS tagging and dependency parsing with beam search. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 110–114. Association for Computational Linguistics, Sofia, Bulgaria, **2013**.

[100] Miguel Ballesteros and Bernd Bohnet. Automatic feature selection for agenda-based dependency parsing. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 794–805. Dublin City University and Association for Computational Linguistics, Dublin, Ireland, **2014**.

[101] Yue Zhang and Stephen Clark. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 562–571. Association for Computational Linguistics, Honolulu, Hawaii, **2008**.

[102] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, **2016**.

[103] Ryan McDonald and Fernando Pereira. Online learning of approximate dependency parsing algorithms. In *11th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, Trento, Italy, **2006**.

[104] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. **2014**.

[105] Rami Al-Rfou', Bryan Perozzi, and Steven Skiena. Polyglot: Distributed word representations for multilingual NLP. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 183–192. Association for Computational Linguistics, Sofia, Bulgaria, **2013**.

[106] Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. Punctuation: Making a point in unsupervised dependency parsing. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, pages 19–28. Association for Computational Linguistics, Portland, Oregon, USA, **2011**.

[107] Ahmet Üstün, Murathan Kurfalı, and Burcu Can. Characters or morphemes: How to represent words? In *Proceedings of The Third Workshop on Representation Learning for NLP*, pages 144–153. Association for Computational Linguistics, Melbourne, Australia, **2018**. doi:10.18653/v1/W18-3019.

[108] Xuezhe Ma and Eduard Hovy. Neural probabilistic model for non-projective MST parsing. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 59–69. Asian Federation of Natural Language Processing, Taipei, Taiwan, **2017**.

[109] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., **2019**.

[110] Travis Oliphant. NumPy: A guide to NumPy. USA: Trelgol Publishing, **2006–**. [Online; accessed ¡today¿].

[111] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50. ELRA, Valletta, Malta, **2010**. `http://is.muni.cz/publication/884893/en`.