Åbo Akademi

Biomedical Text Dependency Parsing with the Neural Turku Parser

Åbo Akademi University
Faculty of Science and Engineering
Department of Information Technology
Thang Ngo Minh
Supervisors: Sampo Pyysalo, Luigia Petre
June 2020

ÅBO AKADEMI UNIVERSITY
Department of Information Technology

THANG NGO MINH: Biomedical Text Dependency Parsing with the Neural Turku Parser

---

English abstract

Syntactic analysis (parsing) is an important Natural Language Processing task that is required in many textual analysis methods in the biomedical domain. This thesis presents the approach taken by me as a member of the TurkuNLP group in the CRAFT Structural Annotation 2019 shared task, a task on dependency parsing. The main system of the approach is the Turku neural parser, a dependency parser originally developed by the TurkuNLP group that achieved great results on previous dependency parsing tasks. I considered and evaluated a variety of strategies to adopt the parser to the biological domain, including the incorporation of both pre-trained and newly made word embeddings, a combination with the other in-domain textual corpora, a modified version of the co-training technique, and taking advantage of information from named entity recognition. In the end, the TurkuNLP team achieved the highest results on the test set of the CRAFT shared task with a labeled attachment score of 89.695%. Future efforts could be focused on fully converting the CRAFT data into universal dependency standard and integrating contextual word representations into the parser to continue improvement upon the parser performance.

Keywords: natural language processing, BioNLP, deep learning, dependency parsing

Acknowledgement

I would like to thank my main supervisor Sampo Pyysalo for his guidance on the writing of my master's thesis and my 2019 summer internship with the Turku NLP Group. The whole project has been genuinely challenging and he truly helped me out considerably in my learning journey despite his busy work schedule.

I would like to thank Luigia Petre, my second supervisor in Åbo Akademi who has the responsibility to grade my thesis. I also thank Marina Walden who advised me on how to manage the thesis project where I as an Åbo Akademi student has a main supervisor from the University of Turku.

Changing the topic of my thesis midway was a tough decision to make for me, thus I thank Paavo Nevalainen, my former thesis supervisor who guided me for a while and then introduced me to Sampo when I expressed the desire.

And finally, I would like to thank my family and friends, both people in my home country Vietnam who always support and encourage me from 8000 kilometres away, and my friends in Finland who make living and studying abroad a joyful experience.

In Turku, June 2020

Thang Ngo Minh

TABLE OF CONTENTS

1 Introduction

The main topic of this thesis is syntactic analysis, also known as parsing, an important Natural Language Processing (NLP) task, and more specifically, a type of parsing called dependency parsing (the other type is constituency parsing). Computational parsing has a very long research tradition, and while it is usually not seen as a stand-alone application, it is an essential building block for many other NLP tasks, for example machine translation (Galley and Manning, 2009) and dialogue system (Sugiyama et al., 2013). In the biomedical domain, many methods working with biomedical text benefit greatly from parsing, for instance information extraction techniques are frequently built on finding the shortest syntactic path to determine the relationships between mentioned entities in the text (Luo et al., 2016).

The CRAFT Structural Annotation (SA) Task 2019 is a shared task competition on dependency parsing, using the biomedical CRAFT corpus as its textual source. The goal is for the participating systems to parse the unannotated unseen test data as accurately as possible. In the summer of 2019, I had the privilege of joining the TurkuNLP team based in the University of Turku's NLP group as a participant in the competition. Relying on the Turku neural parser previously developed by the group, our team considered and evaluated a number of strategies to best adopt the domain-agnostic parser into biomedical domain, especially for this particular CRAFT corpus. After many experiments performed in a relatively short development time, we finally came out victorious, as our three submitted models achieved the highest results out of all of the competitors, which, in a sense, means

accomplishing state-of-the-art performance. This thesis is an in-depth exploration into the project, digging deep into the inner working of the Turku neural parser as well as the various techniques employed, both failed and succeeded ones, along with an analysis of the results. I hope that this thesis can be a part, however small, of the continuously growing effort of applying deep learning techniques into NLP tasks.

The rest of the thesis is structured as follow. Chapter 2 provides the necessary background of the project: the definitions of dependency parsing, its research history both in general language and in specialized biomedical domain, as well as an overview of the CRAFT SA task. Chapter 3 dives into the theoretical framework behind the Turku neural parser, our primary system, along with important machine learning concepts such as word embedding and co-training. Chapter 4 presents and analyzes many of those same methods during the development phase, finished with an investigation of the final submitted results. Chapter 5 discusses some of the less fruitful ideas that while they did not work out very well for this project might still be useful for other related endeavours, together with some predictions into the future. Finally, chapter 6 concludes the thesis by summarizing the main findings and some recommendations as a call for action.

2 Background

2.1 Dependency Parsing

A very important topic in Natural Language Processing(NLP) is parsing, also known as syntax analysis or syntactic analysis. This refers to the process of finding the correct syntactic structure of a string of symbols, for example a sentence, conforming to a given formalism or given grammar. Two of the most popular formalisms are Phrase Structure Grammar and Dependency Grammar.

2.1.1 Phrase Structure Grammar

While similar formalism and less formal notions of phrases had existed before (Bloomfield and Hockett, 1933), Phrase Structure Grammar is usually thought to have been introduced by Noam Chomsky (Chomsky (1957)), the influential American linguist who is sometimes called ”The father of modern linguistics”. It is based on the constituency relation instead of the dependency relation of the dependency grammar. Figure 2.1 shows a parse tree based on phrase structure grammar.

The way constituency parsing works is that it breaks a sentence into its constituents which are called phrases. Constituents are then themselves broken down into smaller constituents, such as noun phrase, verb phrase, preposition phrase and so on. The recursive nature of the phrasal structure makes it easy to embed long sentences, theoretically even ones composed of infinitely long phrases.

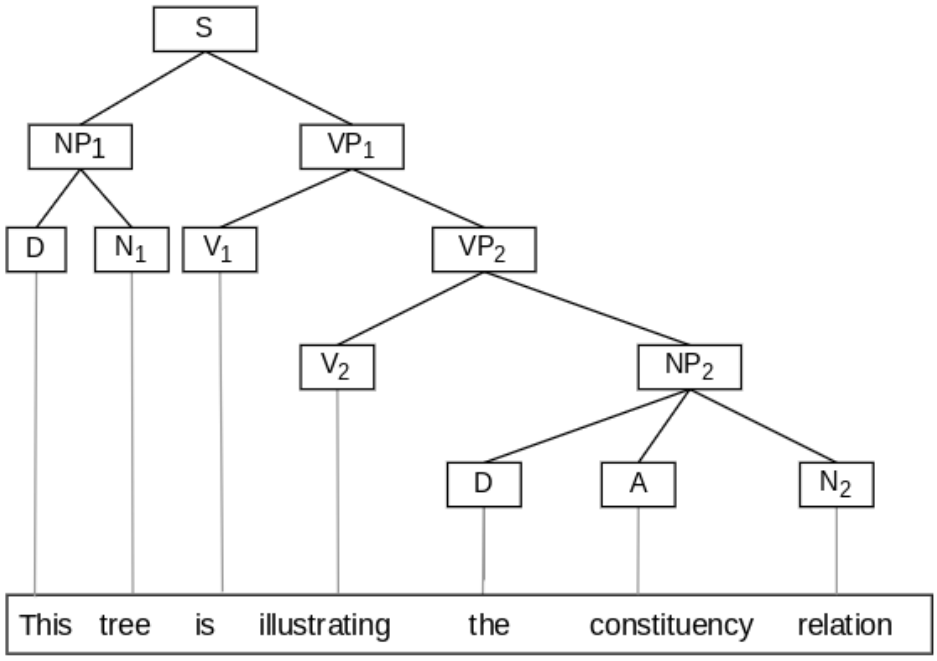

Figure 2.1: Constituency-based parse tree example

2.1.2 Dependency Grammar

.

A very distant type of grammar compared to phrasal structure grammar, dependency grammar does acknowledge phrases, although it lacks phrasal nodes in its structure. Instead dependency grammar sees the verb as the center of the clause structure, and all other syntactic units, or words, are either directly or indirectly connected to the verb via dependencies, which are directed links. The theory of dependency grammar can be traced back to the work of Lucien Tesnière, a prominent French linguist(Tesnière et al. (1959)).Figure 2.2 shows how a short sentence can be parsed based on dependency grammar:

The grammatical relations between individual linguistics units or words are the point of interest in Dependency Grammar. The directed links point from the ”head” words (or governors) to the words (or dependents) that modify those ”head” words. For example, in figure 2.2, the arrow moving from *is* to *hearing* indicates that *hearing* modifies *is*, and the label *sbj*, or *subject* assigned to the arrow describes the nature of this dependency, here

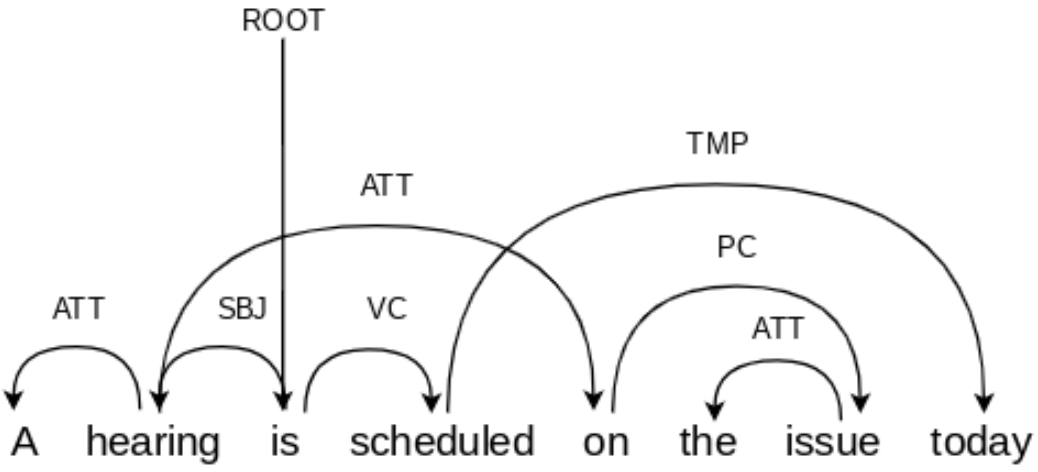

Figure 2.2: Dependency-based parse tree example

hearing is the subject of the verb *is*.

The difference between dependency and phrase structure grammars comes largely from the initial division of the sentence. Based on the analysis of clause structure from the works of Noam Chomsky, the phrase structure parsing first splits the clause into a subject noun phrase (NP) and a predicate verb phrase (VP). Lucien Tesnière, on the other hand, puts the verb as the root of all clause structures. From his viewpoint, the subject-predicate division comes from term logic, an outdated branch of logic originated from the Greek philosopher Aristotle and thus should be excluded from linguistics (Tesnière et al. (1959): 103-105). A pretty interesting philosophical point on how languages work can be made here: if one accepts the view that all clauses can be divided syntactically into subject-predicate, then one might prefer the phrase structure grammar, while if one rejects this division, then one must consider the verb as the root of all clause structures(as every clause needs to have a verb to be considered a clause), and one might go down the path of dependency grammar.

For most of the history of NLP, phrase structure grammar was the more popular approach with a large proportion of work specifically on the Penn Treebank (Marcus et al., 1993). In the contemporary NLP landscape though, parsing based on dependency gram-

mar has been utilized more frequently. This usually stems from practical concerns: dependency grammar is readily applicable to languages with free word order (like Russian, Finnish) as shuffling words around does not change the sentence structure, and potentially more readily used in downstream applications. Dependency parsing was chosen for the CRAFT 2019 Structural Annotation Task and will be the focus of this thesis.

2.2 Related Work

2.2.1 Dependency Parsing For General Texts

Parsing has an enormously long history and has always been an important part of linguistics even in the earliest days. As far back as the 5th year Before Common Era (BCE), Pāini, an ancient Sanskrit philologist and grammarian considered ”the father of linguistics”, tried to describe the grammatical structure of the Sanskrit language in his book Aādhyāyī (Pāini and Katre (1987)). Many of his syntactic rules look very similar to the dependency relations of modern times. In the first millennium there were plenty of works on Arabian grammars and dependency relation remained their basic approach. Ibn Madā, the Arab Muslim polymath from Spain, is treated as the first linguist to address the term *dependency* in a grammatical sense that we use today. For that reason arguably the dependency grammar concept precedes the phrase structure grammar by centuries (Percival, 1990). The idea of context-free/constituency grammars is basically a modern invention, dated back to Noam Chomsky's work as mentioned before.

The first dependency parser was probably developed by David Hays, one of the founders of computational linguistics in 1962 (Hays, 1962). From then, dependency parsing systems were increasingly being built to parse natural languages. However until the 1990s, treebank, an indispensable component of contemporary parsing, was not widely used. Through out those decades, people would write out explicit grammar rules, hand-build the parsers based on the grammars, and then let them parse raw text. The idea was that

having humans specify the rules beforehand that can theoretically speaking capture an infinitive number of possible phrases would prove to be efficient. However in practice this idea did not turn out very well, and thus the rise of annotated treebanks. They completely revolutionized computational linguistics as a whole and dependency parsing in particular.

At first, it seemed like building a treebank was much slower as it is a fairly menial work and much less useful than building a high-level grammar. However a treebank gives us many other benefits. First, treebanks are very reusable. Before treebanks, everyone who tried to build a parser would invent their own notations of grammar rules, and those became more and more complex as time went by to the point where they cannot be utilized by other people's parsers, therefore it was very difficult to share work between people. A treebank can be reused for multiple purposes, not only for dependency parsing but also for other tasks like part-of-speech tagging. In addition to that, linguists can also benefit from treebanks, for instance they usually look for different language constructions in their respective treebanks. Second, treebanks serve as valuable data if we want to apply machine learning methods into parsing. Annotated data provides the frequencies, the distributional information and the context of the dependency relations, all are very valuable in helping the machine learning systems learn the hidden patterns in the data. This is especially powerful for ambiguous sentences, as they might have different interpretations from different parsers. Having the right grammatical structures in context will help machine learning models learn those structures. Finally, treebanks provide an easy way to evaluate parsing systems. The first large-scale treebank published is the Penn Treebank (Marcus et al. (1993)), and its Wall Street Journal section was the focus of parsers for many years.

There have been many methods of building a dependency parser and I will just try to mention some of the most prominent ones.

Karlsson et al. (1995) advocates Constraint Grammar, an example of the eliminative approach, where grammar is viewed as a set of constrains. Parsing is therefore a constraint

satisfaction problem which can be solved by continuously eliminating constraint-violate analyses until only valid analyses remain. Eisner (1996) proposes a dynamic programming algorithm with complexity $O(n^3)$ for dependency parsing and 3 contrasting probabilistic models based on it.

There was a big problem for parsers before the decade 2000s: the speed of the parsers or to be more precise, their time complexity. In many parsers, after parsing each word, we have to decide what to do next: for example whether to shift left-arc or right-arc. The exhaustive search of exploring every possible parse exponentially explodes the complexity considerably. The dynamic programming approach reduced this to cubic time and could explore most of the possibilities and therefore it was the mainstay in those decades. In the decade 2000s, we see the application of machine learning into parsing, and this was a breakthrough as it improved the complexity to linear time while sacrificing only a fraction of accuracy. The machine learning classifiers will predict the next best action to take, and in the simplest form, there is absolutely no searching. Yamada and Matsumoto (2003) train a support vector machine for their deterministic bottom-up parser, meaning the algorithm does not backtrack and it focuses on the small level details first before determining the high level structure of a sentence. Nivre and Scholz (2004) uses a hybrid bottom-up/top-down linear-time parsing algorithm guided by memory-based (or instance-based) classifiers. McDonald et al. (2005) implement online large-margin multi-class training (Crammer and Singer (2003)), a method where the classifier is updated after each single data point instead of after training the whole batch, and it aims to maximize the margins or the minimum distances from the data points to the decision boundary. This method is built on top of a graph algorithm, where a Minimum Spanning Tree was created for each sentence. Nivre et al. (2006) introduce Maltparser, a greedy discriminative transition-based bottom-up parser also guided by machine learning classifiers (memory-based learning and support vector machine mentioned above).

In October 2012, Krizhevsky et al. (2012) introduced a deep convolutional neural

network (CNN) that won the big ImageNet competition by a large-margin compared to traditional machine learning systems, signifying the beginning of the deep learning revolution. Constituency parsing saw development in neural networks before dependency parsing, with the earliest notable attempt was from 2004, (Henderson, 2004). Chen and Manning (2014) was the first successful application of neural network into dependency parsing. It was motivated by the 3 problems that the conventional feature representations suffer from: sparsity (the features, particularly lexicalized features are very sparse), incompleteness (even with expert-involved handling, features still did not match every useful word combination) and expensive computing (in their experiments, conventional parsers spend up to 95% of their time generating indicator features). They built a neural network directly on the stack and buffer configuration, then represent all words, POS tags and dependency labels as dense vectors with a cube activation function.

After this, neural network approaches took the dependency parsing world by storm, and like with most deep learning approaches, simply adding larger and deeper networks with better tuned hyperparameters would do wonder for the results. In addition to that, Weiss et al. (2015) reintroduced searching into the game with a beam search decoding to learn the final neural layer. Beam seach is breadth-first search with the exception that at each level only a predetermined number of best states, called beam depth, are stored and expanded.

More recently, Dozat and Manning (2016) revived the graph-based dependency parser in the neural world with a bidirectional long short term memory (LSTM) network combined with deep biaffine classifiers. Dozat et al. (2017) adapted the system into the CoNLL 2017 shared task on Universal Dependency parsing and achieved great results (con, 2017). I will explore in detail their approach in **Section 3. Methods** as the Turku neural parser was built based on it.

2.2.2 Dependency Parsing For Biomedical Texts

With the rapid growth of biomedical literature, especially in PubMed[1] (only the abstracts) and PubMed Central[2] (full text articles), the task of cataloging the essential research results requires sophisticated automation. Thus information extraction has become a key focus in the biomedical NLP community. Many works in biomedical relation extraction emphasize the significant importance of syntactic information (Airola et al. (2008)), Björne et al. (2011), Liu et al. (2013), Luo et al. (2017), Peng et al. (2017)).

The considerable differences between in-domain texts and general English texts naturally lead to the idea that parsers's performance is also domain-dependent. Miwa et al. (2010) compares multiple parsers and shows that training on biomedical sources with syntactic analysis is important to achieve state-of-the-art performance. A few annotated corpora have been produced to support training and evaluating the NLP systems on biomedical publications, for example GENIA (Kim et al. (2003)) and Penn BioIE (Kulick et al. (2004)) which are constituency treebanks, and BioInfer (Pyysalo et al. (2007b)) which follows the Link Grammar formalism. Link Grammar is similar to dependency grammar, however its links do not need to have directions, thus link grammar doesn't describe the head-dependent relationships (Sleator and Temperley (1995)). Pyysalo et al. (2007a) later converted the BioInfer corpus into the Stanford Dependency representation(de Marneffe and Manning (2008)), making it the first dependency-based annotated biomedical corpus. Verspoor et al. (2012a) introduced the Colorado Richly Annotated Full Text (CRAFT) corpus, which will be elaborated in **Section 2.3** as it is the object of interest in this thesis.

A number of native dependency parsers for biomedical domain were developed, for example Pro3Gres (Schneider et al. (2004)), combining a hand-written dependency grammar and a lexicalized probability model, and Gdep (Sagae and Tsujii (2007)) or Genia dependency parser, a version of KSdep dependency parser trained on the GENIA tree-

[1]https://www.ncbi.nlm.nih.gov/pubmed/

[2]https://www.ncbi.nlm.nih.gov/pmc/

bank, which uses the LR algorithm (Knuth (1965)) alongside with ensemble techniques. Many of these similar early efforts were analysed and compared in Miyao et al. (2008). Howevertheir application was fairly limited, and basically before the deep learning revolution, the main workhorse of biomedical dependency parsing was the approach described in McClosky and Charniak (2008). They self-train the standard C/J parser (Charniak and Johnson (2005)), a constituency-based parser to improve upon it. Self-train is a semi-supervised learning approach where an existing model is used to soft-label some extra data, then include this extra data into the training labeled data and train a second model, this can be repeated. After self-training, the results would be combined with heuristic conversions into dependency relations like the Stanford dependencies.

A popular used tool for biomedical parsing is the Stanford parser mentioned above, which is domain-agnostic. Nevertheless, it can be applied directly into in-domain data with great results (Agarwal and Yu (2010), Ananthakrishnan et al. (2013)). Little effort has been spent on developing deep learning fueled dependency parsers exclusively for biomedical domain, though some tried to extend the Stanford parser with specialist lexicons (Wang et al. (2015)). Zhang et al. (2019) compares 4 neural dependency parsers on clinical texts, including the Stanford parser, the Bist-parser (Kiperwasser and Goldberg (2016a), the dependency-tf-parser (Kiperwasser and Goldberg (2016b)), and the jptdp parser (Nguyen et al. (2017)) and concludes that: (1) all parsers are greatly improved after retraining on large annotated clinical treebanks, signifying a need for sophisticated in-domain corpora and (2) somewhat surprising, the word embeddings for initial features generated from general texts can have the same performance with the ones created from clinical texts.

2.3 CRAFT 2019 Structural Annotation Task

2.3.1 Overview

The CRAFT Structural Annotation Task (SA) is one of the tasks offered as part of the CRAFT Shared Tasks 2019 (ST)[3]. The tasks take advantage of the CRAFT corpus as the data for all of the tasks. This corpus consists of 97 full text journal articles selected from the Mouse Genome Informatics curation pipeline. The articles were manually annotated for a wide variety of language phenomena, spanning structure, semantics, and coreference.

The CRAFT SA task requires the participants to develop an automatic-parsing system that can generate dependency parses of sentences in full-length articles from the CRAFT corpus. For the most part, CRAFT SA follows the conventions of the previously held dependency parsing shared tasks in the Conference on Computational Natural Language Learning (CoNLL) 2017 and 2018 (Zeman et al. (2017), Zeman et al. (2018)).

To stay as close to real-world scenarios as possible, these tasks provide only the plain text articles as the test data instead of already tokenized and tagged text. In addition to syntactic analysis, the participating systems also need to perform sentence segmentation, tokenization, part-of-speech tagging, lemmatization, and morphological features identification. What makes the CRAFT tasks different is the focus on biomedical domain texts instead of the more general texts from the CoNLL. In addition, CRAFT is unique among syntactically annotated biomedical corpora in that its texts are drawn from full-text articles, rather than only article titles and abstracts. Its use of full-text articles makes the CRAFT corpus much more valuable than many other syntactically annotated biomedical corpora.

[3] https://sites.google.com/view/craft-shared-task-2019/home

2.3.2 Data

The dependency structures that serve as development and evaluation data are not themselves manually curated. Instead they are directly translated from the manually curated constituency parses. The modified Penn Treebank (PTB) formalism (Verspoor et al. (2012b)) was used for the constituency parsing, and then the dependency parses were automatically generated from the Penn treebank data with the help of the NLP4J library[4]. The methodology of converting from treebank data to dependency structure is described in Choi and Palmer (2012). Note that this step only transforms the source PTB data into the CoNLL-X format (Buchholz and Marsi (2006)), which is an older format using the Stanford Dependency (SD) representations. The CoNLL-X data would then be converted into CoNLL-U[5] format, a revived version of the CoNLL-X using the Universal Dependency (UD) representations. Unfortunately, this final conversion addressed only the format, not the representation especially the dependency representation - the most important part of the data. This resulted in a data structure which did not follow the UD standards completely and led to some quite peculiar problems in my development process. This point will be expanded on later.

The development data for this task consists of the publicly released dependency parses of sentences in the 67 CRAFT articles. They are in the CoNLL-U format. The testing data consists of dependency parses of sentences in 30 CRAFT articles that was only publicly released after the final evaluation period. All of these 97 articles have been annotated in exactly the same ways.

[4] https://emorynlp.github.io/nlp4j/

[5] https://universaldependencies.org/format.html

2.3.3 The CoNLL-U Format

The segment below shows an example sentence with its dependency parses in the CoNLL-U format.

```
# sent_id = 12607
text = These mice overexpress mutant APP from a vector that can be regulated by doxycycline.
1       These   these   DET     DT              2       det     _       _
2       mice    mouse   NOUN    NNS     Number=Plur     3       nsubj   _       _
3       overexpress     overexpress     VERB    VBP     Tense=PresIVerbForm=Fin 0       root    _
4       mutant  mutant  NOUN    NN      Number=Sing     5       compound
5       APP     app     NOUN    NN      Number=Sing     3       dobj
6       from    from    ADP     IN              3       prep    _
7       a               DET     DT              8       det     _
8       vector  vector  NOUN    NN      Number=Sing     6       pobj
9       that    that    DET     WDT     PronType=Int,Rel        12      dep             8.re
10      can     can     VERB    MD      VerbType=Mod    12      aUX     _       _
11      be      be      VERB    VB      VerbForm=Inf    12      auxpass _       _
12      regulated       regulate        VERB    VBN     Aspect=PerflTense=PastIVerbForm=Part    8       relcl
13      by      by      ADP     IN              12      agent           _
14      doxycycline     doxycycline     NOUN    NN      Number=Sing     13      pobj    _       SpaceAfter=No
15                      PUNCT   .       PunctType=Peri  3       punct   _
```

The first 2 comment lines store the sentence id (treebank-wide) and the full raw text of the sentence. A sentence can consist of one or more word lines, each word lines contain the following 10 fields (columns):

1. ID: The word index, an integer. Starting from 1 for each sentence
2. FORM: The word form itself or a punctuation symbol for a punctuation
3. LEMMA: The lemma or stem of the word,which is the base or dictionary form of the word. For example: *mouse* is the lemma of *mice*
4. UPOS: The universal part-of-speech tag. For example: *from* has the tag ADP (adposition, a cover term for prepositions and postpositions)
5. XPOS: The language-specific part-of-speech tag. It is an underscore if not available. In the CRADT data they were taken from the Penn Treebank part-of-speech tags. For example: *mice* has the tag NNS, meaning it is a plural noun

6. FEATS: A list of the morphological features taken from the universal feature inventory or from a defined language-specific extension. It is an underscore if not available. For example, *VerbForm=Inf* means the word is an infinitive verb

7. HEAD: The head of the current word, which is either a value of ID (the governor word that this word modifies) or zero (0) if it is the root. For example: *can* has its HEAD = 12, thus its head is the 12 indexed word, *regulated*

8. DEPREL: The universal dependency relation to the HEAD (it is the root if HEAD = 0) or a defined language-specific subtype of one. For example: *can* is the AUX (auxiliary) of *regulated*

9. DEPS: The enhanced dependency graph in the form of a list of head-deprel pairs

10. MISC: Any other annotation

2.3.4 Evaluation Metrics

Following the CoNLL'18 shared task, the evaluation of the CRAFT SA task used the same evaluation script[6]. The dependency parser performance would be evaluated on the following 3 metrics Zeman et al. (2018):

LAS: Labeled attachment score The percentage of nodes that are correctly assigned both the syntactic head and the dependency label (the HEAD and DEPREL fields). All nodes are considered in the evaluation, including punctuations. The metric will take word tokenization mismatches into account. Therefore a dependency is scored as correct only if both nodes of the relation match existing gold-standard nodes.

MLAS: Morphology-aware labeled attachment score Similar to LAS, with additional requirements of having certain selected features predicted correctly. The metric is

[6] `https://universaldependencies.org/conll18/evaluation.html`

evaluated only on content words, discarding function words and punctuation. The selected features include certain morphological features, hence the name morphology-aware.

BLEX: Bi-lexical dependency score Similar to MLAS, it focuses on the relations between content words. Howeverit brings lemmatization into the evaluation instead of the morphological features. Therefore35e343 the parents must be aligned, the universal parts of their relation types must be identical and must be listed as "content relations" (same as in MLAS), and their lemmas must match.

3 Methods

3.1 General concepts of neural networks

3.1.1 Artificial neural network

Artificial neural networks are machine learning models that are inspired by the structure of the biological neural networks that form animal and human brains. The idea was first proposed by McCulloch and Pitts (1943), and after decades of development, now in our modern time with the increased computing power from distributed computing and graphic processing units (GPU), large networks with complex architectures have become popular. Because these networks usually have multiple layers, deep learning is the name for this subfield of machine learning. Figure 3.1 shows the structure of a simple neural network.

Neural networks usually have the form of a weighted, directed graph. The nodes of the networks are artificial neurons, which are simulated neurons. They are mathematical functions that receive one or more weighted inputs, sum them up and then pass the sum through an activation function to produce an output. The activation functions are usually non-linear, for example the sigmoid function has a S-shaped curve and squashes the values to between 0 and 1.

3.1.2 Recurrent neural network

Recurrent neural network (RNN) is a class of artificial neural network specifically designed for sequence problems. The first RNN is thought to be the Hopfield network

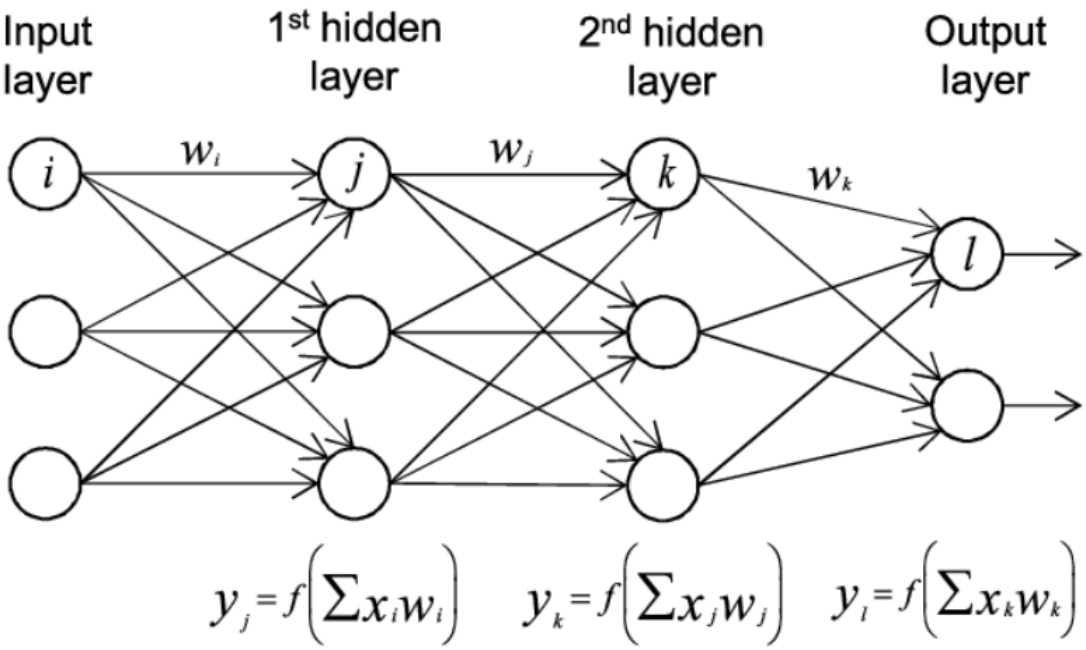

Figure 3.1: Structure of a simple neural network. Source: Vieira et al. (2017)

(Hopfield, 1982). The architecture of a simple RNN is shown in figure 3.2

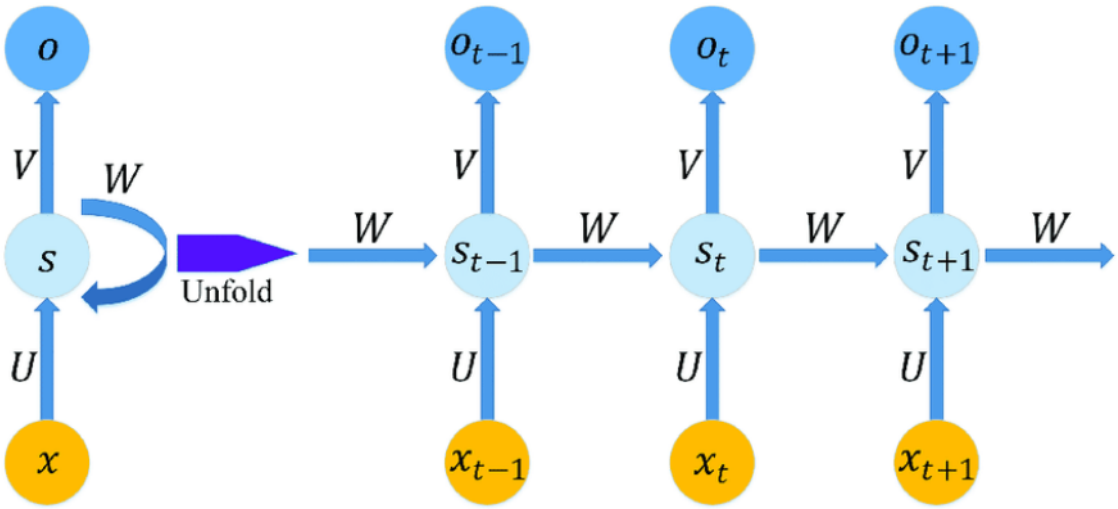

Figure 3.2: A simple recurrent neural network and its unfolding architecture. U, V and W are the weights of the hidden layer, the output layer and the hidden state, respectively. Source: Bao et al. (2017)

A RNN can be conceived as a feed-forward neural network with the addition of loops.

The neurons can not only pass their signal forward but also laterally. This enables each neuron to produce its own hidden state that acts like a memory of the previous neuron, thus the sequential information of the input can be retained. The hidden state is calculated from the current input and the previous hidden state.

One big problem that larger RNNs meet is the vanishing gradient problem, defined in Hochreiter's diploma thesis (Hochreiter, 1991). Simply put, when backpropagation moves from the last layer to the initial layer, the derivatives are multiplied down the network by the chain rule. If the hidden layers use an activation function that squashes a large input range into a small output range (like the sigmoid function), the produced derivatives are small, and this makes our gradient greatly reduced the further we propagate down the initial layers. As a consequence, the weights and biases of the initial layers will not be updated efficiently in each training session, leads to the inaccuracy of the entire network.

To address this problem, a special kind of RNN was invented.

3.1.3 Long short-term memory

LSTM was explicitly designed to learn long-term information, avoiding the vanishing gradient problem. There are many variants of its architecture, in figure 3.3 is a typical structure of a LSTM unit or memory cell. The gates regulate the information that goes into the cell state, and their activation function is usually the sigmoid function, a "S" shape function that squashes everything into the range between 0 and 1. The forget gate decides to what extent a certain value should be forgotten by the cell, the input gate controls what new information will be added to the cell state, and finally the output gate handles how much of the information in the cell state might be computed in the output activation of the LSTM memory cell. The output activation function in RNN is commonly the tanh function, which pushes the output values in the interval [-1, 1].

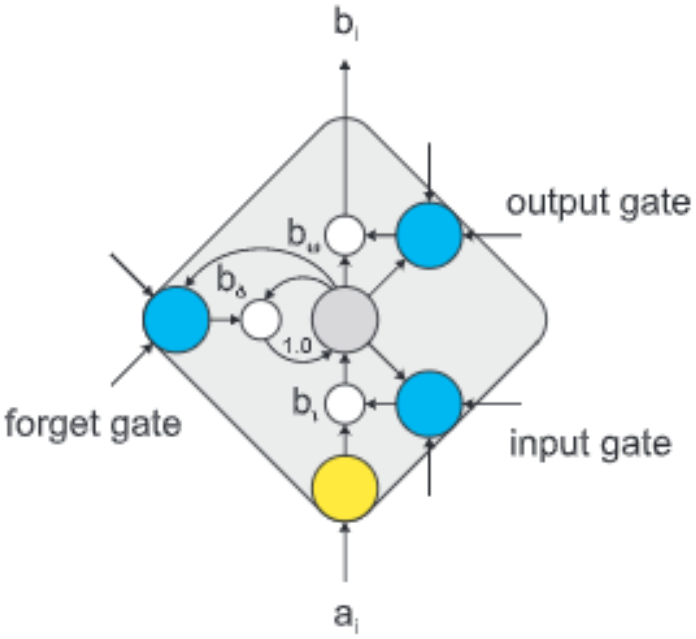

Figure 3.3: A LSTM memory cell. Source: Sundermeyer et al. (2012)

3.1.4 Gated Recurrent Unit

GRU was introduced by Cho et al. (2014a), and, like its more complex predecessor long short term memory (LSTM) (Hochreiter and Schmidhuber (1997)), its objective was to solve the vanishing gradient problem that standard recurrent neural networks suffer from. GRU networks utilize update gates to decide how much past information should be stored, and forget gates to gauge how much should be filtered out, making it a very powerful architecture. Figure 3.4 compares the architectures of standard RNN with GRU networks.

3.1.5 Bidirectional Recurrent Neural Network

Bidirectional RNN, invented in 1997 by Schuster and Paliwal (1997), splits the hidden neurons in a standard RNN into 2 directions, forward and backward. This way the output layer can receive input information from both the past and the future of the current time frame. Figure 3.5 depicts the general structure of RNN and bidirectional RNN.

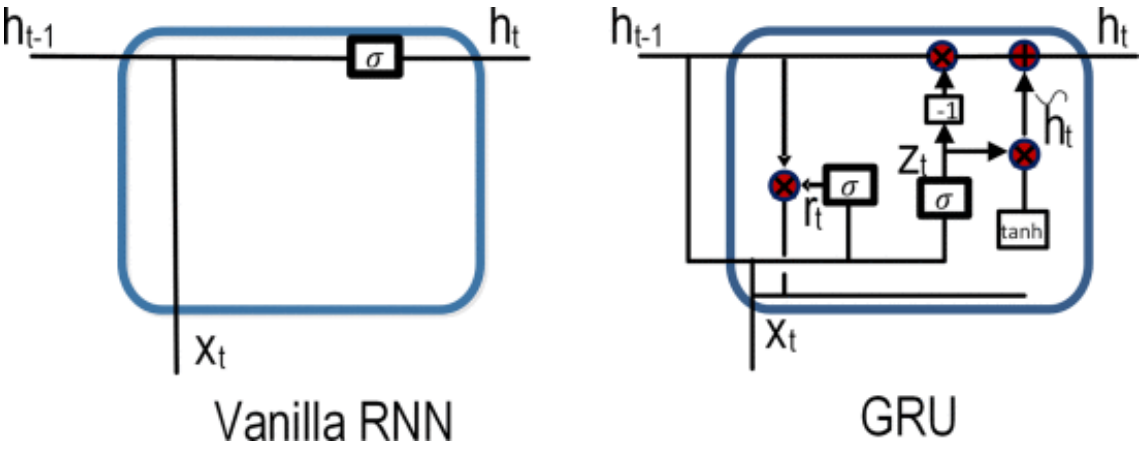

Figure 3.4: Illustration of vanilla RNN and GRU units. Source: Zhao et al. (2018)

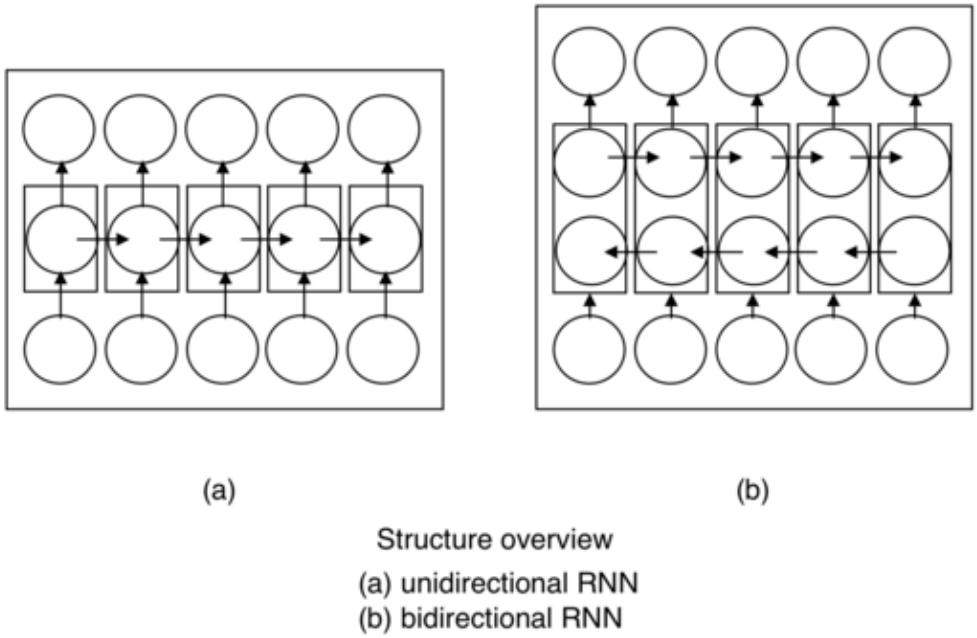

Figure 3.5: Illustration of vanilla RNN and bidirectional RNN. Source: Schuster and Paliwal (1997)

3.1.6 Attention

The first type of attention mechanism, called addictive attention was developed by Bahdanau et al. (2014) for their neural machine translation system. Their paper ”Neural Machine Translation by Jointly Learning to Align and Translate” has become a seminal

paper in the deep learning world as attention is basically omnipresent in most of the state-of-the-art models in recent years. The basic structure of an attention model is the attention of Figure 3.6.

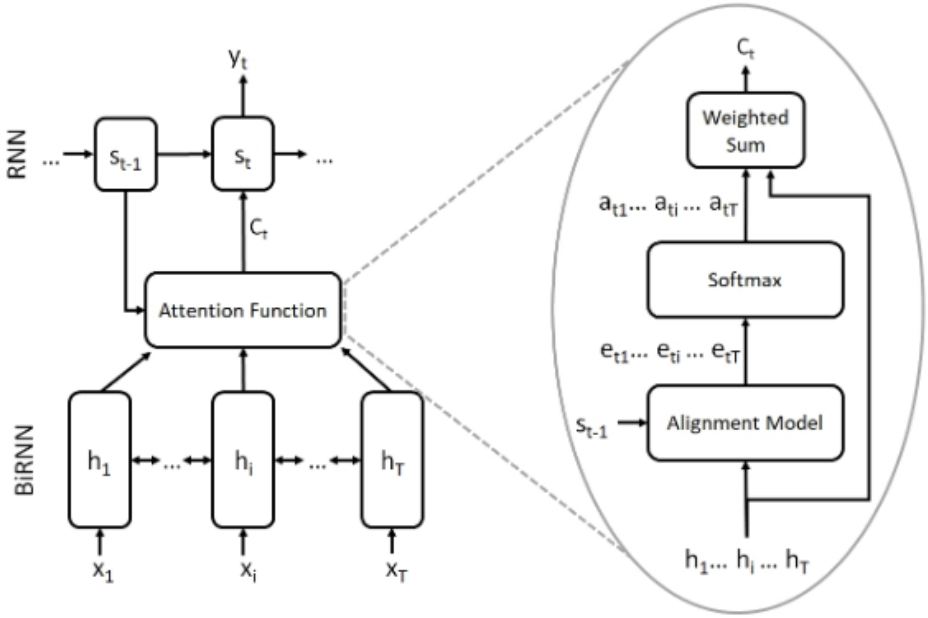

Figure 3.6: The attention model in a RNN network architecture. Source: Galassi et al. (2019)

Attention's main function is to calculate a weight distribution of the input sequence in order that more relevant elements have higher weights, and its output is called a context vector, which is a weighted combination of all of the input states. In this way, for each new unit, it can receive information not only from just the last unit's state, but also the most essential information from all of the input states.

3.1.7 Input-feeding Attention

Figure 3.7 shows how the input-feeding attention mechanism works.

Introduced in Luong et al. (2015), the input-feeding attention tries to address the limit of word alignment in a normal attention mechanism, which is the task of identifying which words are translation of which words between 2 texts. Basically, in Neural Machine

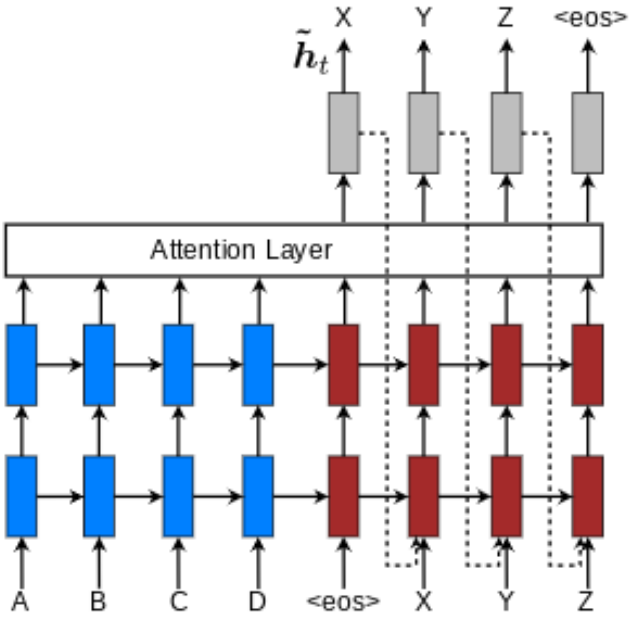

Figure 3.7: The structure of the input-feeding attention model. Source: Luong et al. (2015)

Translation (NMT), the alignment decision of the current time-step should bear in mind the past alignment decisions. This is similar to how in a non-neural machine translation or manual translation, a coverage set is needed to track the already translated words. Howevernormal attention decisions are made independently. In their proposed approach, the attentional vector or the output of the previous decoder will be fed as input into the next decoder, instead of just being used in the attention calculating of that decoder. This will provide information about past alignments to the model.

3.1.8 Encoder-Decoder

First introduced by Cho et al. (2014b) and made popular by Sutskever et al. (2014) at Google, the encoder-decoder model's power lies in its ability to map sequences of output and input with different lengths, a very common problem in machine translation as different languages use different number of words for the same sentence, or different number

of characters for the same word. The encoder is a 2-layer bidirectional LSTM that receives as its input the token as a sequence of single characters along with that token's morphosyntactic features and produces encoding vectors. These encoding vectors, combined with the input-feeding attention mechanism will help the decoder, which is a 2-layer unidirectional LSTM, generates the sequence of output characters. Figure 3.8 illustrates the general architecture of an encoder-decoder model.

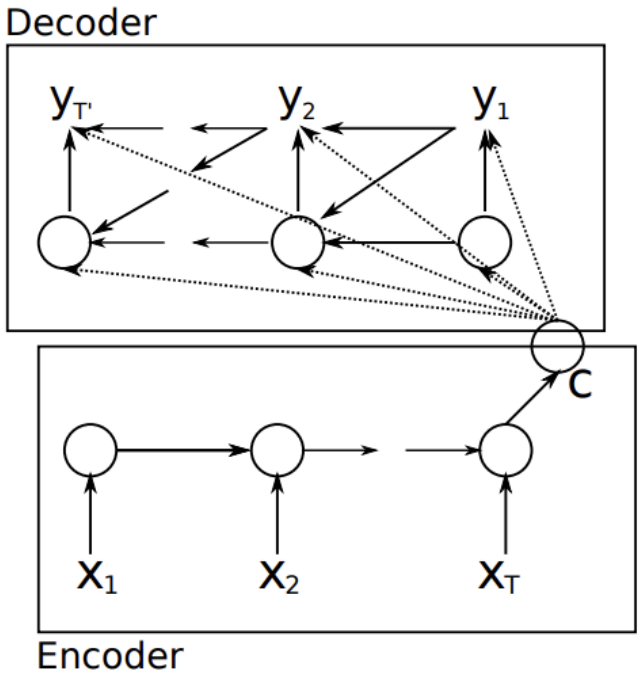

Figure 3.8: The architecture of the encoder-decoder architecture. Source: Cho et al. (2014b)

3.2 The Turku Neural Parser Pipeline

The Turku Neural Parser Pipeline[1] (Kanerva et al. (2018)) is the main parser used to participate in the CRAFT SA 2019 shared task. It is an end-to-end pipeline which comprises 4 main components: segmentizer, POS/morphological tagger, lemmatizer, and depen-

[1]https://turkunlp.org/Turku-neural-parser-pipeline/

dency parser, The design goals of the parser are:

Easy to use Docker images are available to make the parser runable without installing all of the dependencies. If installed manually along with the dependencies, a few command lines are enough to start training and parsing. The pipeline's components and hyperparamters are available for modifying in configuration files.

Reusable All of the components are packed into a single software and are easily customized. The parser can work with most natural languages, with pre-trained models for more than 50 of them. It is also label agnostic, thus the treebanks do not need to adhere to the UD guidelines.

High performance The parser should be able to achieve state-of-the-art or very close to state-of-the-art results on all four tasks, for any language (even for languages with low-resource treebanks). For this reason, the components for segmentation, tagging and parsing were heavily relied on previously available works with modifications only when it proves fruitful. However, as it was deemed that the performance of the previous lemmatizer was undesirably weak, an original approach to lemmatization was created.

To summarize, the segmentation is carried out by UDPipe (Straka and Straková (2017)), the tagging and parsing is performed by a modified version of the graph-based parser by (Dozat et al. (2017)), and finally, the lemmatization makes use of the OpenNMT neural translation toolkit (Klein et al. (2017)). In the following subsections, I will describe each of these units in detail. Because the parser and the tagger of Dozat et al. (2017) have almost identical architecture, and the focus of this thesis is on dependency parsing, I will provide descriptions for the inner working of the parser before the tagger, even if technically the tagger comes first in the pipeline, as the parser makes use of the POS tags - the output of the tagger - as one of its features.

3.2.1 Tokenizer

In NLP, a text has a multileveled structure: a *document* (a file, an article) comprises one or multiple *paragraphs*, a paragraph is built from one or more *sentences*, and a sentence is basically a sequence of *tokens*. Normally a token is a *word*. Sometimes a token can be composed of multiple syntactic words, or it can be a punctuation character. For most of the languages, the Turku neural parser simply performs these tasks (dividing a document into separate sentences, each sentence a sequence of individual tokens) by adopting the already excellent system of UDPipe. In a nutshell, sentence splitting and tokenization are jointly predicted by a single-layer bidirectional gated recurrent unit (GRU) network.

The goal of the tokenizer in UDpipe is to divide an input segment into individual sentences and tokens, this can be reformulated as a classification task: we need to classify each input character into one of the three classes: token boundary follows (the end of a token), sentence boundary follows (the end of a sentence), and no boundary. This is done by running two GRU with reversed order on the same input segment (hence the name bidirectional GRU), concatenate the forward and backward hidden states produced by the two GRU. This concatenated one will then be put into a softmax function to normalize it into a probability distribution in the interval $(0, 1)$. The tokenizer is trained by an Adam optimizer (Kingma and Ba (2014)).

3.2.2 Parser

Dozat et al. (2017) have a rather sophisticated word-embedding scheme that needs to be explained separately first. The embedding of each token is a sum total element-wise of three different representations. The first is a pre-trained word embedding, usually from external sources, and can be easily plugged into the parser. The second is a ”trained” embedding consisting of words that appear at least twice in the training dataset, called ”holistic frequent word embedding”. And finally, an embedding that is built up from its sequence of the word’s characters. This character-level model is illustrated in Figure

3.2.2.

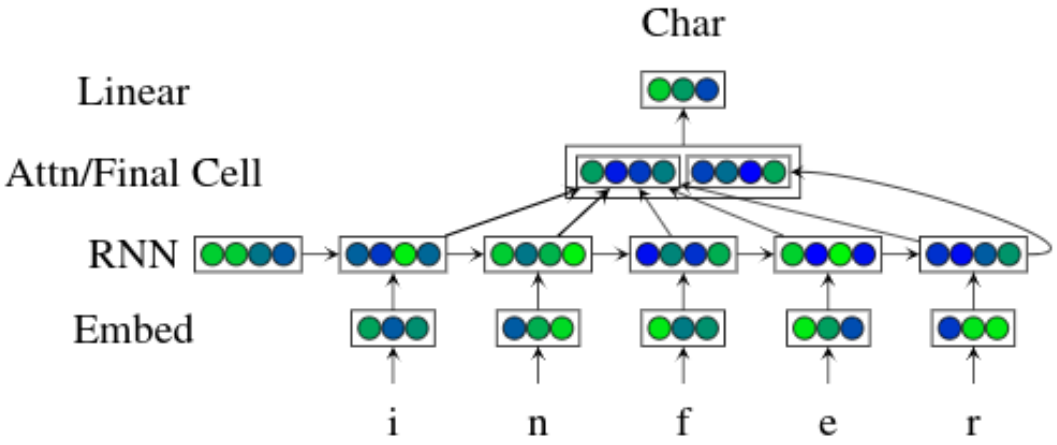

Figure 3.9: The architecture of the character-level word embedding by Dozat et al. (2017) . Source: Dozat et al. (2017)

Each character of a token is converted into a trainable vector, and all characters of this token will become the input of an unidirectional LSTM network.

After feeding the sequence of character embeddings into the LSTM network, it produces a sequence of recurrent states, and these states need to be combined and converted into one single embedding vector for the original token. Two approaches are considered, and in the end both are combined together. First is to simply take just the last recurrent state. In a a sequence-to-sequence (seq2seq) network like this, the final memory cell would store all of the important information accumulated through all of the characters. We just need to transform this into the desired dimensionality of the token embedding. Second is a more complex approach using attention mechanisms proposed by Cao and Rei (2016).

Their idea was to run an attention model on all of the hidden states generated by the LSTM model, this should create a context vector which is a representation of the input sequence. Dozat et al. (2017) combine this context vector with the last state as stated

above, and perform a linear transformation to get the final character-level embedding of the original token. Finally, they sum element-wise this character-level embedding with the pre-trained embedding and the holistic frequent word embedding to get the final token embedding of a given token.

These word embeddings will be furthermore concatenated with their corresponding tag embeddings (from the tagger) to create the final input ready for dependency parsing. We now turn our attention to the architecture of the parser of Dozat et al. (2017) itself. Figure 3.2.2 is an overview graphical representation of this parser.

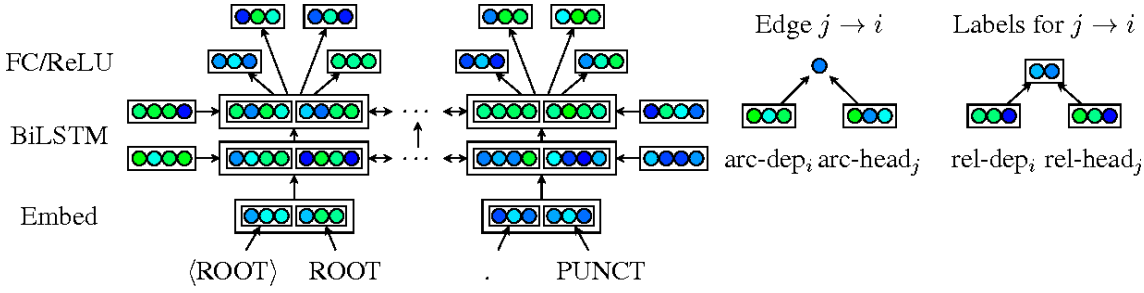

Figure 3.10: The architecture of the neural parser by Dozat et al. (2017)

. Source: Dozat et al. (2017)

The basic architecture was built off from the Bist-parser of Kiperwasser and Goldberg (2016a), which was also developed from McDonald et al. (2005), one of the first graph-based parser. Here is a walkthrough of the process of parsing by the parser:

Bidirectional LSTM The input to the model is a sequence of token embeddings and their corresponding POS tags. These are fed through a bidirectional LSTM network, where the network will learn to create context-aware token representations. For the next block, only the hidden state or the output state of the final LSTM layer is kept, the cell state is excluded.

ReLU layers Four separate ReLU (rectifier linear unit, which outputs only non-zero input values) layers run through the context-aware token representations, creating four

different vectors. The purpose of the 4 smaller ReLU layers is to reduce unnecessary information for an individual decision. Basically, for each token, what we are trying to predict are not just 1, but 4 different categories: its head, its dependents, the label of the edge from the token to its head, and the labels of the edges from its dependents to the tokens. The recurrent state or the token's context-aware representations of course has to store all of this information which would be very redundant when we try to compute just one category, making the parser more prone to overfitting and slows the parser down. Having 4 multilayer perceptron (MLP) ReLU reducing the dimensionality and applying nonlinearity transformation can help us avoid these 2 troubles.

The first vector is for the token as a dependent looking for its head. Conversely, the second vector is for the (same) token as a head looking for all of its dependents. The third vector is for the (same) token as a dependent determining its label. And vice-versa, the fourth vector is for the (same) token as a head determining all of the labels of its dependents. Each original token will have their own version of these four vectors now, and all of these are the input for the final block

Biaffine classifiers The 2 separate biaffine classifiers receive the four type of vectors as input[2]. An **affine** transformation has the form of $f(\mathbf{x}) = A\mathbf{x} + \mathbf{b}$, if the bias b is a zero vector then it is called a linear transformation. A **biaffine** transformation will be a variation of applying an affine transformation twice, thus in the form of $f(\mathbf{x}) = HA\mathbf{x} + \mathbf{b}$

The first classifier receives the first and second type of vector, produces a score for each pair of tokens. For each token, the pair that has the highest score will be its most likely head. The second classifier receives the third and fourth type of vector, produces a score for each label of all of the pairs token-the token's head. For each relation, the label with the best score will be the most likely label of this relation. After this, the 2

[2]For the mathematical notations, I follow the following conventions: lowercase italic letters for scalars and indices, lowercase bold letters for vectors, uppercase italic letters for matrices, and uppercase bold letters for higher order tensors

biaffine classifiers are again jointly trained with the goal of optimizing the sum of their cross-entropy losses (A loss function that measures the performance of a classifier whose output is a probability value between 0 and 1 like softmax).

Below is, again, the entire parsing structure put formally:

1. **Token Embedding**

   Given as input a sequence of *m* character embeddings $(\mathbf{v}_1^{char}, ..., \mathbf{v}_m^{char})$, the LSTM network with the initial state $\mathbf{r}_0$ will produce recurrent state $\mathbf{r}_i$. This recurrent state consists of the hidden state $\mathbf{h}_i$ and cell state $\mathbf{c}_i$ and we need to extract them.

$$\mathbf{r}_i = LSTM(\mathbf{r}_0, (\mathbf{v}_1^{char}, ..., \mathbf{v}_m^{char})_i) \tag{3.1}$$

$$\mathbf{h}_i, \mathbf{c}_i = split(\mathbf{r}_i) \tag{3.2}$$

   The linear attention then run through the stack of all of the hidden states *H*.

$$\mathbf{a} = softmax(H\mathbf{w}^{(attn)}) \tag{3.3}$$

$$\tilde{\mathbf{h}} = H^T\mathbf{a} \tag{3.4}$$

   Finally this will be concatenated with the final cell state $\mathbf{c}_m$

$$\tilde{\mathbf{v}} = W(\tilde{\mathbf{h}} \oplus \mathbf{c}_m) \tag{3.5}$$

   The final token embedding $\mathbf{v}^{(word)}$ is the sum element-wise of this character-level vector $\tilde{\mathbf{v}}$, the pre-trained embedding $\mathbf{v}^{(pre)}$ and the holistic frequent word embedding $\mathbf{v}^{(hol)}$

$$\mathbf{v}^{(word)} = \tilde{\mathbf{v}} \oplus \mathbf{v}^{(pre)} \oplus \mathbf{v}^{(hol)} \tag{3.6}$$

2. **Bidirectional LSTM**

   The input is a sequence of *n* word embeddings $(\mathbf{v}_1^{(word)}, ..., \mathbf{v}_n^{(word)})$ and their *n* corresponding POS tags $(\mathbf{v}_1^{(tag)}, ..., \mathbf{v}_n^{(tag)})$. These pairs will be concatenated into one

stack of *n* vectors $(\mathbf{x}_1, ..., \mathbf{x}_n)$ and then fed into the BiLSTM with initial state $\mathbf{r}_0$

$$\mathbf{x}_i = \mathbf{v}_i^{(word)} \oplus \mathbf{v}_i^{(tag)} \tag{3.7}$$

$$\mathbf{r}_i = LSTM(\mathbf{r}_0, (\mathbf{x}_1, ..., \mathbf{x}_n) \tag{3.8}$$

$$\mathbf{h}_i, \mathbf{c}_i = split(\mathbf{r}_i) \tag{3.9}$$

3. **ReLU Perceptron Layers**

   Using the four ReLU layers on each of the hidden state h_i, four different vectors are generated

$$\mathbf{h}_i^{(arc-dep)} = \mathbf{MLP}^{(arc-dep)}(\mathbf{h}_i) \tag{3.10}$$

$$\mathbf{h}_i^{(arc-head)} = \mathbf{MLP}^{(arc-head)}(\mathbf{h}_i) \tag{3.11}$$

$$\mathbf{h}_i^{(rel-dep)} = \mathbf{MLP}^{(rel-dep)}(\mathbf{h}_i) \tag{3.12}$$

$$\mathbf{h}_i^{(rel-head)} = \mathbf{MLP}^{(rel-head)}(\mathbf{h}_i) \tag{3.13}$$

4. **Deep Biaffine Classifiers**

   The first biaffine classifier involves with the task of computing *n* score vectors $\mathbf{s}_i^{(arc)}$ by applying biaffine transformation on the $\mathbf{h}^{(arc-head)}$ and $\mathbf{h}^{(arc-dep)}$ vectors.

$$\begin{aligned}\mathbf{s}_i^{(arc)} = H^{(arc-head)} W^{(arc)} \mathbf{h}_i^{(arc-dep)} \\ + H^{(arc-head)} \mathbf{b}^{T(arc)}\end{aligned} \tag{3.14}$$

   The most likely head word *j* of word *i* is predicted as the word *j* with the highest score $s_{ij}^{(arc)}$

$$y_i^{/(arc)} = \arg\max_j s_{ij} \tag{3.15}$$

   The traditional affine transformation has the form of $W\mathbf{h} + \mathbf{b}$, here both W and $\mathbf{b}$ are transformed by $H^{(arc-head)}$ first, hence the name **biaffine**. Equation 3.14 has 2 terms. $H^{(arc-head)} W^{(arc)} \mathbf{h}_i^{(arc-dep)}$ describes the probability of word *i* having word *j* as its head based on the information in both $\mathbf{h}^{(arc-head)}$ and $\mathbf{h}^{(arc-dep)}$ vectors. While the second term, $H^{(arc-head)} \mathbf{b}^{T(arc)}$ describes the same probability based on only the $\mathbf{h}^{(arc-dep)}$ vector.

The second biaffine classifier is responsible for putting a label to the dependency from a dependent word *i* to its predicted head word $y_i^{/(rel)}$. This time, the score vectors $\mathbf{s}_i^{(rep)}$ is computed by affine transform the $\mathbf{h}^{(rel-head)}$ and $\mathbf{h}^{(rel-dep)}$ vectors.

$$\begin{aligned}\mathbf{s}_i^{(rep)} &= \mathbf{h}_{y_i^{/(arc)}}^{T(rel-head)} U^{rel} \mathbf{h}_i^{(rel-dep)} \\ &+ W^{(rel)}(\mathbf{h}_i^{(rel-dep)} \oplus \mathbf{h}_{y_i^{/(arc)}}^{(rel-head)} \\ &+ \mathbf{b}^{rel}\end{aligned} \tag{3.16}$$

The most likely label *j* of the dependency from word *i* to its predicted head word $y_i^{/(arc)}$ is predicted as the label *j* with the highest score $s_{ij}^{(rel)}$

$$y_i^{/(arc)} = \arg\max_j s_{ij} \tag{3.17}$$

The equation 3.16 has 3 terms. The first term $\mathbf{h}_{y_i^{/(arc)}}^{T(rel-head)} U^{rel} \mathbf{h}_i^{(rel-dep)}$ describes the probability of discovering a label based on the information in both the $\mathbf{h}^{(rel-head)}$ and $\mathbf{h}^{(rel-dep)}$ vectors. The second term $W^{(rel)}(\mathbf{h}_i^{(rel-dep)}$ describes the same probability based on either of the $\mathbf{h}^{(rel)}$ vectors. And finally, the bias $\mathbf{b}^{rel}$ describes the prior probability of discovering a label. The 2 biaffine classifiers are jointly trained by minimizing the sum of their cross-entropy losses.

3.2.3 Tagger

POS tagging is the task of assigning to a word a particular POS tag, which describes generally the word's grammatical role in the sentence. This task is of extreme importance as its outputs, the UPOS, the XPOS tags (and in the Turku pipeline, also the morphological tags) are essential inputs for the parser and the lemmatizer.

As mentioned above, the tagger in the Turku Neural Parser is a modified version of the tagger of Dozat et al. (2017), and the architecture of the tagger is almost identical with the parser's structure. In brief, the tagger also receives as its input the triple embeddings of the tokens, the pre-trained, the holistic word, and the character-level representations,

has its first component as a Bidirectional LSTM network, and then 2 separate ReLU classifiers to generate vectors for the 2 types of tag, the universal part-of-speech UPOS and the language-specific part-of-speech XPOS. Then, 2 affine classifiers are also trained jointly with the goal of optimizing their sum of cross-entropy losses, predicting the most probable POS tag for the token.

So, formally speaking, given a sequence of *n* tokens ($\mathbf{v}_1^{(word)}, ..., \mathbf{v}_n^{(word)}$), a BiLSTM with initial state $\mathbf{r}_0$ runs through them and produces hidden state $\mathbf{h}_i$ for each token.

$$\mathbf{r}_i = BiLSTM(\mathbf{r}_0, (\mathbf{v}_1^{word}, ..., \mathbf{v}_n^{word})_i) \tag{3.18}$$

$$\mathbf{h}_i, \mathbf{c}_i = split(\mathbf{r}_i) \tag{3.19}$$

The ReLu will then runs through the hidden vectors $\mathbf{h}_i$ to create 2 representations for 2 kinds of POS tag.

$$\mathbf{h}_i^{(Upos)} = \mathbf{MLP}^{(Upos)}(\mathbf{h}_i) \tag{3.20}$$

$$\mathbf{h}_i^{(Xpos)} = \mathbf{MLP}^{(Xpos)}(\mathbf{h}_i) \tag{3.21}$$

An affine classifier computes a score vector $\mathbf{s}_i^{(Upos)}$ for each token *i*. The chosen tag will have the highest score.

$$\mathbf{s}_i^{(Upos)} = W^{(Upos)}\mathbf{h}_i^{(Upos)} + b^{(Upos)} \tag{3.22}$$

$$y_i^{/(Upos)} = \arg\max_j \mathbf{s}_{ij}^{(Upos)} \tag{3.23}$$

Another distant affine classifier perform the same actions for the Upos tags.

$$\mathbf{s}_i^{(Xpos)} = W^{(Xpos)}\mathbf{h}_i^{(Xpos)} + b^{(Xpos)} \tag{3.24}$$

$$y_i^{/(Xpos)} = \arg\max_j \mathbf{s}_{ij}^{(Xpos)} \tag{3.25}$$

A modification to the tagger was made in the Turku Neural Parser: the tagger can now predict not only the **UPOS** and **XPOS** tags but also the morphological features. A simple but ingenious trick is employed: on the annotated treebanks, at the training stage, the

FEAT column is concatenated into the **XPOS** column, making them into one long strings, and therefore forcing the tagger into learning and predicting these 2 fields together. Therefore every unique combination of XPOS tags and morphological features will become a single class to be learned, and after assigning a predicted class for a raw token, the class will be split back into the original columns. This straightforward approach actually can lead to great results, as explained in Kanerva et al. (2018), it not only outperformed the Turku team's original more complex method and achieved the 3rd rank in morphological features, it also improved the LAS score as it seemed like the parser benefited from learning the morphological features. Thus for the Turku parser, the tag embeddings $\mathbf{v}_i^{(tag)}$ consists of 2 elements, the UPOS vectors and the XPOS vectors, with the XPOS vectors represent both the original XPOS tags and the morphological features.

3.2.4 Lemmatizer

Lemmatization is the task of inferring the base form or dictionary form called lemma from a word form. They could be the same, for example the lemma of *the* is no other than *the*, while the lemma of *loves* is *love*. Lemmatization is performed independently from dependency parsing and also was not evaluated in the CRAFT SA 2019 shared task. Lemmatization could still be evaluated using the evaluation script although it was not taken into account when gauging the parser's performance. Below I will briefly describe the approach taken inside the Turku neural parser pipeline.

The lemmatizer in the Turku pipeline has a novel approach that promises to achieve state-of-the-art performance for most of the languages covered by the UD treebanks. In Kanerva et al. (2019), the authors argue that part-of-speech and morphosyntactic features are enough to solve the lemmatization ambiguity problem so prevalent in the task. Inspired by the top systems in the CoNLL-SIGMORPHON 2017 shared task (Cotterell et al., 2017), which requires the participants to re-inflect or predict the inflected words given their lemmas and morphosyntactic features, the authors cast lemmatization as a

sequence-to-sequence rewrite task, where the model takes as its input the sequence of characters of the word along with the sequence of its corresponding morphosyntactic features (in our case, the **UPOS**, **XPOS**, and **FEAT** columns). Below, the word *mice* will have the following input and desired output:

```
INPUT: m i c e NOUN NNS Number=Plur
OUTPUT: m o u s e
```

When represented like this, the lemmaization task can be understood as a translation problem from the string of the inflected word + its morphosyntactic features to the string of its lemma.Therefore any powerful sequence-to-sequence model could be utilized. The OpenNMT: Open-Source Toolkit for Neural Machine Translation (Klein et al. (2017)) was adopted as the main model. Figure 3.11 depicts the full structure of the model.

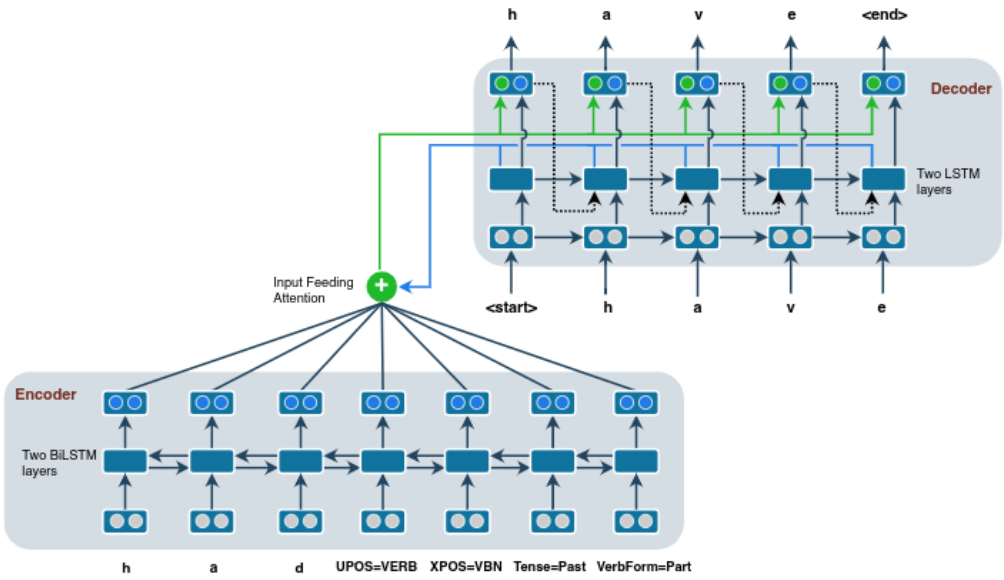

Figure 3.11: The encoder-decoder model architecture of the lemmatizer. Source: Kanerva et al. (2019)

In the Turku pipeline, the lemmatizier is put as the last component, after tagging and parsing, as the tagger and parser do not need the lemmas as input, while the lemmatizier requires the POS tags and morphological features predicted by the tagger as its input.

Because the tagger reads the entire sentence at once, it can learn about the context of each word, and the morphosyntactic tags it generates will have sufficient information to help the lemmatizier with ambiguous words.

3.3 Word Vectors

As mentioned above, the parser receives as its input a triple representations of the tokens, two of them, the holistic frequent word embeddings and the character-level embeddings, are produced by the parser itself, while the third embedding, the pre-trained vectors, are simply plugged in to initiate the training. A number of them were tested during the experiments, as will be described in **Section 4. Results**. In this section, I will briefly detail the theory behind word embeddings and some of the techniques that were used to induce new word vectors for the parser.

3.3.1 Word Embeddings

To enable computers and later machine learning models to work with raw text, we need to transform them into the only form they can understand, namely numbers. One of the traditional ways of representing word is one-hot encoding, where the vector representations of words have only one element as 1, all the other elements are 0. They are sparse vectors and the length of the vector is equal to the number of unique words in a corpus. Though simple to understand and implement, there are 2 huge problems: we could not capture the relationship between words as the distances between vectors, and the sparseness of the vectors.

Word embedding is the efficient solution. Word embeddings are vector representations of raw words, usually in the form of vectors of real numbers. Abstractly speaking, it can be thought of as a mathematical embedding from a very high dimensional ”word” space to a much lower dimensional vector space. It is the de facto approach to make raw text

understandable and input-ready for NLP models.

The theory behind word embedding has its root in linguistic, mainly in the idea populated by Firth (Firth (1957)):

```
a word is characterized by the company it keeps!
```

Similar to people where we can get to know the personality of a person by getting to know their social circle, words that are used in similar context will have similar meaning. This is called the **distributional hypothesis** (Harris). Word embedding aims to capture this by making similar words have similar representations, or minimizing the dot products of their respective vectors in the vector space. There have been multiple methods of learning word embeddings from text data, and for the CRAFT SA 2019 shared task, 2 main ones were chosen: word2vec and fasttext.

3.3.2 word2vec

Developed by Mikolov et al. (2013) at Google in 2013, word2vec is a neural-network based toolkit for efficient word embedding. Since then, it has become the standard for developing pre-trained word vectors, and many other approaches are just modified versions of word2vec. Word2vec can employ either of the 2 learning models to produce the vector representations: continuous bag-of-words (CBOW) or skip-gram. Figure 3.12 compares these 2 model architectures.

CBOW The CBOW model tries to predict the current word based on its surrounding words or context. Both history and future words are incorporated, thus the word target is the middle one. The word order is not taken into account, hence it assumes a bag-of-word characteristic (where a text is considered a multiset of words, disregarding word order). For example, with a sentence ”He is a smart boy”, the CBOW model will read over the 4 words ”He”, ”is”, ”smart”, and ”boy” to predict the most likely word in the middle, in this case ”a”.

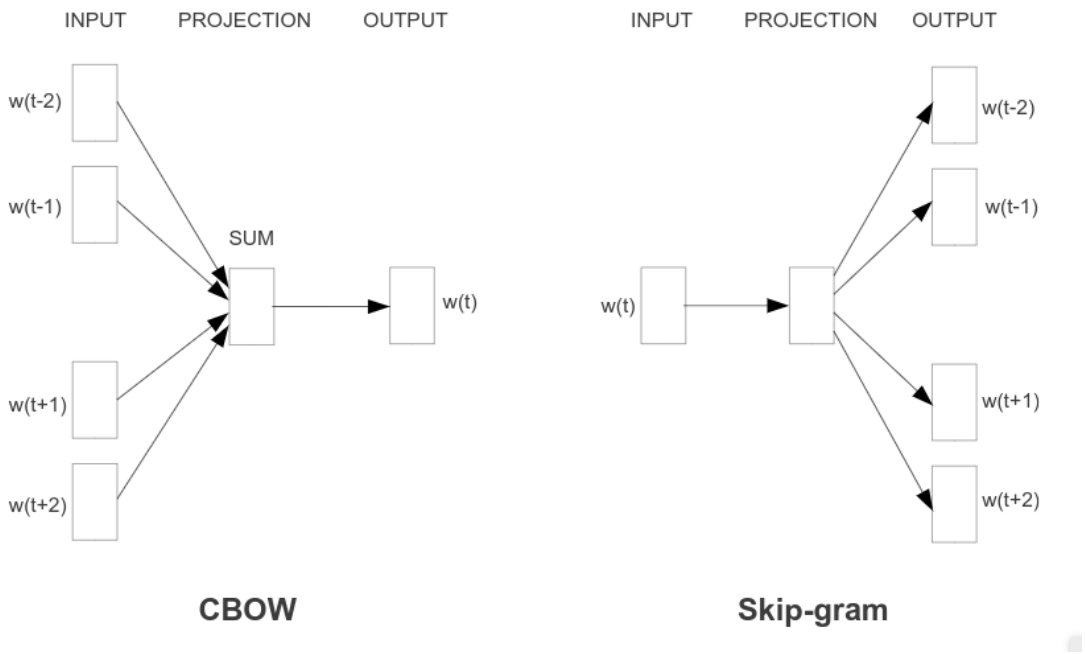

Figure 3.12: The CBOW vs skip-gram models. Source: Mikolov et al. (2013)

skip-gram The skip-gram model is quite similar to the CBOW model, except that it reverses the input and output: it tries to predict the surrounding words based on only the middle word. Again, the center word is utilized, both previous and forward words are predicted. This mirrors a skip-gram analysis, and we can understand the output or context words as a set of 1-skip-n-grams where $\frac{n}{2}$ is the window size. For example, with a sentence ”I love a good course” and current word ”a”, a skip-gram model window size 1 will predict only ”love” and ”good”, while a skip-gram model window size 2 will predict ”I”, ”love”, ”good” and ”course”.

Both types of model focus on the usage context of words, and this context is a window of neighboring words. The window size is a crucial hyperparameter for the models to be chosen carefully as Chiu et al. (2016) shows that training high quality embeddings demand not only good input corpora and model architectures, but also hyperparameter tuning. Mikolov et al. (2013) claims that skip-gram performs better for rare words and CBOW is faster, nevertheless in general their performance is quite similar.

3.3.3 fasttext

Fasttext was proposed by Bojanowski et al. (2016), Joulin et al. (2016) at Facebook in 2016 as an extension of word2vec . Instead of representing a word as one vector, fasttext, based on the skip-gram model, represents each word as a bag of character *n*-gram, each of this character *n*-gram will be transformed into a vector, and the final embedding of each word is the sum of its character *n*-gram vectors. The motivation was that, most methods before fasttext including word2vec only represented the entire words as vectors, not at the character-level, making it hard to have good representations for morphological rich languages, for example Finnish is a highly inflecting language with 15 cases just for nouns. Therefore the *n*-gram vectors will help the model learns the rules of word formation and inflection. Rare words are also better represented as there is a high chance that their *n*-gram subwords are in the corpora, and as such, learnt.

Like word2vec, fasttext can incorporate either skip-gram or CBOW as its learning algorithm. Bojanowski et al. (2016) claims that fasttext is fast to train and outperform other methods that do not take into account subword-level information.

3.4 Co-training

To further improve on the strong result of the combination between the Turku parser and a good word embedding, multiple approaches were considered (will be expanded in section 5 and 6) and most of them did not bear fruit. A modified version of co-training was the only method that pushed the performance further, even just a marginal score. This section briefly summarizes the theory behind co-training. Figure 3.13 demonstrates the approach visually.

In the classical co-training set up, the data is partitioned into two different views, each view will have a different set of features. To ensure the best performance, the two views should be conditionally independent, meaning given the label of an instance, the two

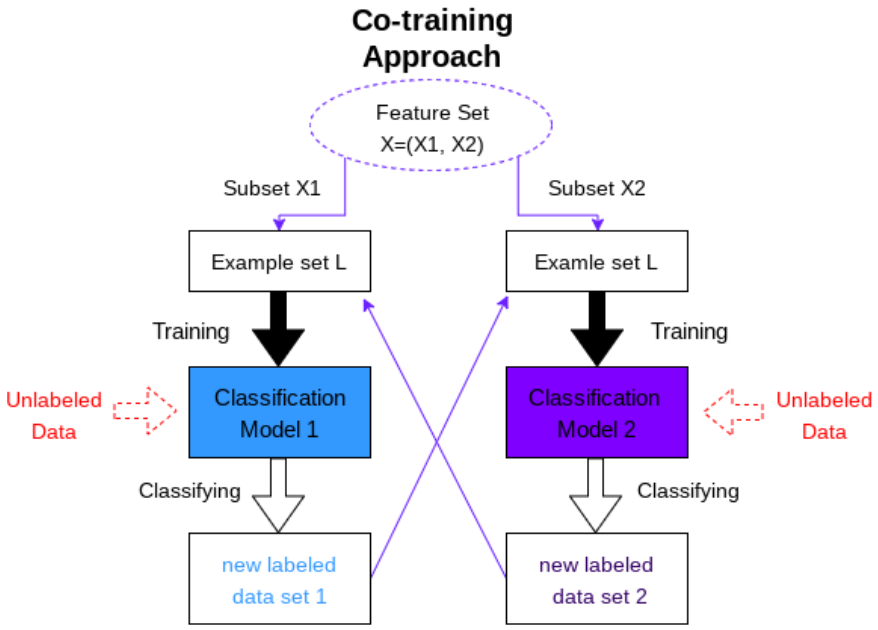

Figure 3.13: The co-training approach

feature sets of this instance are conditionally independent (the likelihood of each feature set provides no information about the likelihood of the other feature set). They should also be sufficient by themselves to let the label of an instance be correctly predicted just from one of the two views. Using the labeled training data, co-training will learn two different classifiers for two views separately. We will then use these two widely different but complementary learners to predict labels for some other unlabeled data. The most confident predictions are then become extra labeled training data, and this step will be iteratively performed.

In the modified version experimented for the CRAFT shared task, the approach was slightly different from the classical co-training approach. There was no feature splitting, the two learners, in this case the Turku parser and UDpipe parser, both learned from the same training data from CRAFT. Another difference is that, in classical co-training approaches, we prefer to have independent classifiers, thus the two classifiers would predict different labels for the same unlabeled instance, they would learn different signals and

make different mistakes. Howeverin this project, the instances that have identical segmentation and tokenization and syntactic structure were included as extra labeled training data, giving us the confidence that they were labeled correctly. The result of this experiment will be analyzed below.

	Train	Test
Documents	67	30
Sentences	21 731	9 099
Tokens	561 032	232 619

Table 4.1: CRAFT Structural Annotation statistics

4 Results

In this section, I will present and analyze the important results of the many methods theoretically explained in section 3 as well as the final results of the test set submissions of the CRAFT SA shared task.

4.1 Data

4.1.1 CRAFT data

For the development of systems of the CRAFT SA shared task, the syntactically annotated CRAFT articles provided by the shared task organizers are the main sources. Table 4.1 outlines an overview of the data

Only the 67 **Train** documents were released publicly for development purpose. The goal was to predict correctly the syntactical structure of the unannotated 30 **Test** documents, and these 30 documents were only published near the submission deadline in the

	Train	Devel	Eval
Documents	47	10	10
Sentences	15 007	3 421	3 303
Tokens	387 473	91 306	82 253

Table 4.2: CRAFT Train data split for development

form of raw text. The annotation procedure of the Train and Test documents were identical. The gold (correct) annotations of the Test documents were only released after the final submissions to help participants in error analysis.

As per the usual recommendations in machine learning projects (Ripley and Hjort (1995), James et al. (2014), Abu-Mostafa et al. (2012)), I randomly split the 67 annotated documents into three different subsets: train, devel and eval. Table 4.2 shows the statistics of this data split.

Using a rough ratio of 70/15/15 percent, 47 train articles were used for training, 10 devel articles were used for regularization via early stopping, and 10 test articles were used for an unbiased evaluation of the learned model. This set-up was kept unchanged for the entire development process to make it easy to compare the performances of different models.

Outside of the provided CRAFT corpus, external sources of data were also utilized, most notable are the pre-trained word vectors (section 3.2), the annotated data for corpus extension, and the unlabeled data used for two purposes: generating new word vectors (section 3.2) and extra training data for co-training (section 3.3). The next sections will be about those three different types of external data.

4.1.2 Pre-trained word vectors

Some readily available high quality pre-trained word embeddings were incorporated into the input of the parser. A reminder that the parser needs three different types of token representations, the character-level embeddings and the holistic frequent embeddings can be computed by the parser itself, the pre-trained embeddings come from outside.

The word embeddings generated by Ginter et al. (2017) were the general embeddings playing the role of the baseline. In the multilingual shared tasks CoNLL 2017 (Zeman et al. (2017)) and CoNLL 2018 (Zeman et al. (2018)), they were provided by the organizers for the participant's English parsing systems. The raw source of English data trained on were the texts gathered from Wikipedia's articles and other sources on the Internet like CommonCrawl, an open repository of free web crawl data [1]. The learning method was word2vec (section 3.2.2) with the following options: skip-gram model, lower-cased text, window size 10 and 100 dimensions.

As noted in section 2.2.2, in-domain biomedical word embeddings usually perform better than general English word vectors for a biomedical focused corpus like CRAFT. Thus some of the previously introduced in-domain word vectors were also taken into radar. Many of them were created by Pyysalo et al. (2013), for example:

Random Indexing word vectors Random indexing, originated from the work of Kanerva et al. (2000), is an approach of building semantic word vectors incrementally. Each word will have an **index vector**, which is a sparse vector full of 0 values, except for a handful of -1 and 1 at random locations. The context vector of a word is produced by scanning through the corpus for all of its occurrences, all of its context windows, and then sum all of the index vectors of its neighbor words. The word vectors were made by using a modified version of an available tool named NADA (unfortunately it is not available

[1] https://commoncrawl.org/

anymore. [2]

word2vec vectors Explained in section 3.2.2, word2vec is an efficient neural-network based method of computing word representations. The tool **word2vec**[3] from Google was utilized to make the new word vectors from unlabeled biomedical data (next section). The following options were chosen: skip-gram model, 200-dimensional vectors and window size 5.

4.1.3 External English data

A large amount of English data was needed to perform the two tasks: creating new word embeddings, and becoming extra training data. Multiple sources were considered, both annotated and unlabeled (raw text):

The Penn Treebank Wall Street Journal section As mentioned in section 2.2.1, the Penn Treebank Marcus et al. (1993) was the first large scale annotated English corpus that was widely used by the parsing community, especially its Wall Street Journal section. The data can be accessed via this link: [4]. They are in Penn Treebank format, with the extension *.mrg*, denoting *merged* which is a merge or combination of the POS tags as well as the syntactical parses. They consist of roughly more than a million tokenized English words.

The GENIA treebank Starting with the work of Kim et al. (2003), the GENIA corpus is the result of a continued effort in providing a comprehensively annotated biomedical corpus for the various tasks of NLP and information extraction (IE). The corpus consists of article abstracts taken from the MEDLINE database of the US National Library of Medicine, which includes references to journal articles in biology focusing on

[2] `http://www.nada.kth.se/xmartin/java`

[3] `https://code.google.com/p/word2vec/`

[4] `https://github.com/scookies/NLP-HW2/tree/master/out/production/NLP-HW2/edu/berkeley/wsj`

biomedicine [5]. The most recent version, version 3.0 has more than 2000 abstracts and more than 400,000 tokens. For the purpose of this thesis I considered a subset of the corpus which was syntactically annotated in the PTB scheme. The corpus in various format can be found here: [6].

The English UD treebanks Like the name implied, Universal Dependencies (UD) [7] is an ongoing project to develop an universal grammatical annotation for all human languages. The UD includes not just syntactical dependencies but also part of speech tags and morphological features. This open community endeavor so far has produced over 100 treebanks for more than 70 languages and they are invaluable resources for those working on parsing technology. Its general annotation structures are based on the modified structures of the Stanford Dependency Representations (de Marneffe and Manning (2008)) (for the syntactical dependencies), the Google part of speech tags (Petrov et al. (2012)) (for the POS) and the Interset interlingua (Zeman (2008)) (for the morphosyntactic features). Currently there are six English UD treebanks [8], for example *UD_English-EWT* or English Web Treebank [9] which contains more than 250,000 tokens.

Unannotated data from Pubmed and PMC Pubmed [10] is an online archive of citations (which include the titles, the abstracts and sometime the links to full-text articles) for biomedical text collected from online books, biomedical journals, and MEDLINE. Currently it has more than 30 millions said citations. PubMed Central (PMC) [11] on the other hand, is a database that stores full text articles from life science and biomedical

[5] https://www.nlm.nih.gov/bsd/medline.html

[6] https://github.com/allenai/genia-dependency-trees

[7] https://universaldependencies.org/

[8] https://universaldependencies.org/en/index.html

[9] https://universaldependencies.org/treebanks/en-comparison.html

[10] https://www.ncbi.nlm.nih.gov/pubmed/

[11] https://www.ncbi.nlm.nih.gov/pmc/

journals. Nearly 6 millions full articles can be accessed for free from the PMC archive.

Instead of using the latest version of these datasets, the 2017 version of the Pubmed baseline distribution and the PMC Open Access dataset were used as the source for generating new word vectors and extra training data as they were readily available from the previous work of my supervisor **Sampo**. The raw text would then be further processed in the following manner: the GENIA sentence splitter [12] split the sentences, then the PTBtokenizer taken from the Stanford Core NLP tools (Manning et al. (2014)) tokenized the sentences, and finally the tokenized sentences were randomly shuffled. In the end, the texts consist of 500 million sentences, which carry 12.5 billion tokens. Now the texts are ready to be incorporated as a source of unlabeled extra training data, or as a source to produce new word embeddings.

To generate new word embeddings from those data sources, readily available tools were utilized, with the 2 main ones are word2vec [13] (section 3.2.2) and fastText [14] (section 3.2.3). As introduced in those sections, the 2 different learning models, skip-gram and continous bag-of-word (CBOW) are both very effective, thus they were both implemented in the 2 tools. As mentioned above, Chiu et al. (2016) points out that the window size is a crucial hyperparameter to the performance of the learning model, thus different values of the window size were tested.

In the next section, I will present the results of some of the most important and successful attempts during the development process when the final test dataset from the organizer was not available.

A note about the evaluation metrics as mentioned in section 2.3.4, the CRAFT shared parsing task had 3 evaluation metrics: LAS, MLAS, AND BLEX. Between these three, LAS (Labeled attachment score) is the most commonly practiced and the most well-

[12] `http://www.nactem.ac.uk/y-matsu/geniass/`

[13] `https://github.com/tmikolov/word2vec`

[14] `https://fasttext.cc/`

established metric. Therefore, I also used LAS as the main goal to optimize during experiments. For the results of the development set, I will report only in terms of their LAS scores. For the final three test set submissions, I will provide not only all three of the metrics but also additional metrics implemented in the CoNLL evaluation script.

4.2 Development Set Results

I experimented with and tested out multiple ideas during the development phase, trying to keep the process in an iterative and incremental manner. Two main strategies proving to be quite effective were the customization of biomedical word embeddings and augmentation of training data.

4.2.1 Word embeddings

To initiate the parser, an external source of word embeddings is needed, and I considered a wide range of them, both general English and in-domain biomedical ones, both pre-trained vectors and newly induced vectors. The main parser used is the Turku parser, with some additional experiments done on UDPipe to act as the baseline. Table 4.3 sums up the most prominent embeddings and their LAS score.

The first notable thing here is that: the parsers can already achieve a very high baseline LAS score of 89.27% (the Turku parser) and 84.66% (the UDPipe parser) utilizing only the general English CoNLL word embeddings. Granted, this is a very high quality embeddings created by the organizers of the CoNLL 2018 shared tasks. After this, the hope was to improve upon this already impressive result using in-domain biomedical word vectors, both pre-made and freshly created. With a clear winner between the Turku parser and UDPiper, I quickly focused on using the Turku parser as the main system, with only a few more experiments performed on UDPipe.

At first, some of the previously introduced word vectors (either created from the previ-

Parser	Word vectors	LAS
Turku	Bio, word2vec/CBOW (window 2)	89.86
Turku	Bio (CRAFT tokens), word2vec/CBOW (default parameters)	89.78
Turku	Bio (CRAFT tokens), word2vec/CBOW (win2)	89.69
Turku	Bio, word2vec/CBOW (default parameters)	89.55
Turku	Bio, word2vec/CBOW (window 20)	89.73
Turku	Bio, FastText/CBOW (default parameters)	89.50
Turku	Bio, word2vec/skipgram (default parameters)	89.63
Turku	Bio, word2vec/skipgram (window 5)	89.34
Turku	CoNLL	89.27
Turku	RI	89.24
Turku	vec-50	88.81
UDPipe	Bio, word2vec/CBOW (window 2)	85.00
UDPipe	CoNLL	84.66
UDPipe	Bio, word2vec/CBOW (default parameters)	84.22

Table 4.3: Development set results with different word embeddings. CoNLL = baseline CoNLL shared task word vectors, Bio = custom word vectors induced on PubMed and PMC articles, CRAFT tokens = input text tokenized with model trained on CRAFT data, RI = Random indexing vectors, vec-50 = 50-dimensional word embeddings from [16]

ous works of my supervisor and his lab or freely available from the NLP community) not only did not improve on the baseline LAS score but they were also lower the score. Two of them are the random indexing vectors and the 50-dimensional vectors with 89.24% and 88.81% respectively. Only the *Bio, word2vec/skipgram (window 5)* vectors previously

created from Pubmed article citations provided the first small improvement in quality: 89.34%.

From then on, the focus was on inducing new in-domain biomedical word embeddings from the large source of raw unannotated text of PMC (full-text articles) and Pubmed (only the abstracts) (seciton 4.1.3). Section 3.2.1 explains all of the details about word2vec, Fasttext, their CBOW and skipgram models as well as their parameters. Different embeddings were created using different settings, and the better options would become narrower based on empirical testing with the development data. Regarding word2vec or fasttext, word2vec was found to be better performed in most of the cases, thus the former became the predominant method of making new word vectors. For the other settings, experimental results indicated that CBOW is the better learning algorithm compared to the skipgram model, where all of the highest scores were achieved with CBOW-induced embeddings. And finally concerning the window parameter, smaller windows were observed to perform better than either the default window or the larger window.

The best performing embeddings scored 89.86% which is a 0.6% higher in LAS accuracy compared to the CoNLL baseline embeddings. This has the following settings: tool word2vec, learning model CBOW, and window size 2. The runner-up embeddings achieved a LAS score of 89.78%, created with the combination of tool word2vec, learning model CBOW, default window size which is 5 for CBOW, and another addition: the raw text was tokenized by a model trained on CRAFT data before becoming the source data to train the word vectors. The idea was to develop a word embeddings that stay as close to the vocabulary of the CRAFT data as possible, minimizing the out-of-vocabulary rate of the vectors. In the final three submissions, two of them used the former embeddings, while the other used the latter embeddings.

4.2.2 Training data augmentation

Singling out the set of word embeddings that performed best on the development CRAFT data was a great progress, after that it is all about enhancing the quality of the training data. A technique called NER (named entity recognition) was considered, however it did not provide any improvements on the data. Increasing the size of training data is another approach, thus a few external source of annotated data were tested to be combined with the original CRAFT data, however they were mostly failed attempts (section 5.1). Only the modified co-training approach (section 3.3) showed a small margin of LAS score increasing. Some selected results from this strategy can be seen in table 4.4.

Parser	Word vectors	Extra data (size, source)	LAS
Turku	Bio, word2vec/CBOW (window 2)	4k sentences, PMC	89.92
Turku	Bio, word2vec/CBOW (window 2)	10k sentences, PMC and PubMed	89.87
Turku	Bio, word2vec/CBOW (window 2)	10k sentences, PMC	89.84
Turku	Bio, word2vec/CBOW (window 2)	6k sentences, PubMed	89.78
Turku	Bio, word2vec/CBOW (window 2)	14k sentences, PMC	89.73
Turku	Bio, word2vec/CBOW (window 2)	3k sentences, PMC	89.64
Turku	Bio, word2vec/CBOW (window 2)	20k sentences, PMC	89.41

Table 4.4: Development set results with extra training data

Here is a detailed account of the strategy: The best parsing models of the Turku parser and UDPipe parser from the previous experiments (trained and tested using only the original CRAFT data) both do the syntactical parsing task on a large source of unannotated data from Pubmed abstracts and PMC full-text articles. Having the parsed results from both models, I compared them to find the sentences that were agreed upon by the two models, both at the level of sentence splitting, tokenization (the sentences have exactly the same words in both models) and at the level of syntax (the words in the sentences

have identical heads and dependencies). I would then randomly sample these sentences in order that they are out of order, varying from sources, different sentence length and so on, making for a helpful source of extra annotated training data. Multiple sets up of this data were combined together with the original CRAFT training data to become the new training data for training new models.

There were some limited increases in the LAS score, with the most optimistic one being the set up with four thousand sentences from PMC full-text articles. Its LAS score 89.92% was only a 0.06% margin higher than the model trained on only the CRAFT data. A potential reason for why this strategy could not work better is that the Pubmed and PMC data covers a wide range of biomedical subjects, while the CRAFT data only focuses on a narrow topic, the Mouse Genome Informatics curation pipeline, making the vocabulary compatibility not that high. Still, the improvement was there though not very substantial, thus I included the best model here as the final submission alongside the 2 previously mentioned CRAFT-data-only models.

4.3 Test Set Results

After the development, every participants could afford 3 of their best models to parse the unannotated raw CRAFT text provided by the organizers near the deadline. They would announce the results shortly after the deadline and also publish the gold data (the correctly annotated test CRAFT data) for error analysis purpose. Table 4.5 summarizes the final test LAS scores of the three runs that I submitted, along with their LAS score during development.

As can be seen, the table actually shows four, not three runs, with an additional post-mortem run. The reason behind this is that, right after I submitted the three runs to the organizers, I realized a mistake that I had made: I did not use the full set of the CRAFT development data for the final model training. As mentioned in section 4.1.1, I divided

Parser	Word vectors	Extra data	LAS(dev)	LAS(test)
Turku	Bio, word2vec/CBOW (window 2)	post-mortem		89.859
Turku	Bio, word2vec/CBOW (window 2)	4k sentences, pmcoa articles	89.92	89.695
Turku	Bio, word2vec/CBOW (window 2)	No	89.86	89.650
Turku	Bio (CRAFT tokens), word2vec/CBOW (defaults)	No	89.78	89.536

Table 4.5: Final submission results on test data

the 67 CRAFT articles provided from the organizers into train-dev-eval sets as the normal standard when developing machine learning systems. The 10 eval articles were very carefully set aside, the trained systems never looked at those articles, preventing the risk of data snooping and guarantee the most objective and non-bias evaluation of the trained systems. However, after determining the best models I should have included those 10 eval articles into the training data and retrain an even more effective model on this larger training data.

The post-mortem work that I have performed does indicate that the inclusion of the 10 eval articles into the training data would produce a model more powerful than all of the three submitted runs with a LAS score of 89.859%. The good news is that, the three submitted models were still the leading champions of the competition. The three runs followed quite closely their development results, with the disparities only 0.3% between the development and test results for all of them, displaying a high generalization degree. While the two models trained only on CRAFT data were very competitive, the run that incorporated the same word embeddings with additional training data from PMC full-text articles proved still be the best, even by only 0.005% above the runner-up.

A more detailed evaluation of the three final submission and the post-mortem run are shown in table 4.6. These are evaluation metrics implemented in the official evaluation

script (seciton 2.3.4).

Metrics	Run 1	Run 2	Run 3	Post-mortem
Tokens	99.593	99.555	99.593	99.592
Sentences	97.590	97.621	97.590	97.423
Words	99.593	99.555	99.593	99.592
UPOS	98.221	98.179	98.184	98.241
XPOS	97.806	97.758	97.789	97.830
UFeats	98.282	98.233	98.265	98.313
AllTags	97.752	97.718	97.729	97.771
Lemmas	98.999	98.981	99.048	99.035
UAS	90.942	90.882	90.794	91.088
LAS	89.695	89.650	89.536	89.859
CLAS	87.373	87.294	87.201	87.524
MLAS	85.549	85.441	85.318	85.688
BLEX	86.630	86.595	86.544	86.841

Table 4.6: Final submission test results for all metrics

The first three metrics concern the tasks of the tokenizer, and here all of the models perform very well: the sentence splitting are over 97%, the token and word boundaries identifiers are all over 99.5%. While English does have contractions which are words made by shortening and combining two different words (for example *don't* is a contraction generated from *do* and *not*), in academic writing their usage is almost always discouraged, thus in the CRAFT data there is no multi-word token, making the tokens and words results identical for each of the model.

The next four metrics evaluate the performance of the tagger: the universal part-of-

speech tags UPOS, the language-specific part-of-speech tags and the morphological features UFeats. We can also see very high results (roughly 98%) for all of the categories.

The lemmatizier also performed very well, with approximately 99% of the word's lemmas predicted correctly, speaking to the effectiveness of the simple but ingenious approach of Kanerva et al. (2019).

Now we come to the final five metrics which gauge the performance of the dependency parsing model in various ways. In brief, the more important metrics are:

UAS how well does HEAD match, or how well just the head word was assigned

LAS how well does HEAD + DEPREL match, thus both the head word and the dependence relation have to be correct

MLAS how well does HEAD + DEPREL + UPOS + UFEATS match, thus not only dependency structure, but also the POS tags and the morphological feature tags need to be correct

BLEX how well does HEAD + DEPREL + LEMMAS match, thus the dependency structure plus the correctly predicted lemma

If we are talking about dependency parsing alone, LAS provides the clearest measure, and here all models achieved over 89.5% accuracy, with the post-mortem model being very near 90%. The UAS scores are higher than the LAS scores even if only by a very small margin, suggesting that predicting the DEPREL labels was more challenging than recovering the HEAD word, and this should be where we can find room to improve that can lead to an increase in the overall parsing performance. It can also be observed that BLEX scores are always higher than MLAS scores, this means that the lemmatizer also did a better job than the tagger.

Increasing the size of the training data especially data that is very similar to the test data is clearly a good approach to keep in mind, as the post-mortem model outdone the

submitted three models in the majority of the metrics, especially the syntactical-analysis related metrics.]

5 Discussion

In this chapter, I want to focus on discussing some of the ideas and approaches that were not very successful at improving the system, along with some projections on the future of dependency parsing.

5.1 What Did Not Work

In this section I present the ideas that, while definitely promising, did not seem to match very seamlessly with the project during development. They were usually abandoned early without any significant results worth reporting. However, I will still try to explain the ideas and their reasons for failing as clear as possible.

5.1.1 Some pre-trained word embeddings

As mentioned in section 4.2.1, the CoNLL general English word embeddings already provided a strong baseline, and many attempts at using other pre-made freely available word vectors were not very fruitful. The improvement only consistently came as a result of creating new biomedical word vectors trained on large sources of in-domain raw literature. The main reason that this can be contributed to is:

Vocabulary mismatch A limitation of using word embeddings is that, because they are the result of a machine learning model after being trained on a finite source of data, the model will not produce the embedding of a new word that it has not seen in the training

data. These are called OOV (out-of-vocabulary) words, and the higher the rate of OOV words, the more new and less reliable embeddings there are when the parser initiates . Table 5.1 shows the OOV rates of some of the vectors along with their development results.

Word Embeddings	OOV rate	Dev LAS
vec-50	11.66%	88.81
CoNLL	3.23%	89.27
RI	1.76%	89.24
Bio (CRAFT tokens), word2vec/CBOW (window 2)	1.14%	89.78
Bio, FastText/CBOW (default parameters)	0.54%	89.50
Bio, word2vec/CBOW (window 2)	0.54%	89.86
Bio, word2vec/CBOW (window 20)	0.54%	89.73

Table 5.1: OOV rate of the word embeddings regarding the CRAFT training data

The best results were only achieved using the set of embeddings with only 0.54% OOV rate. Thus it would be reasonable that the LAS score might be even higher if the OOV rate reaches zero, however the probability that a set of word embeddings stores all of the vocabulary in a new text is very low. The potential lies in how to handle OOV words. The Turku parser has the ability to accumulate vocabulary from the data we are parsing, make embeddings for previously unseen words and add these new embeddings into the parsing model (Kanerva et al., 2017). Technically, this approach eliminates all OOV words.

5.1.2 Corpora incompatibility

Extending the training data with high quality and similar samples is a very reliable way of improving the model, especially in the era of deep learning models. Before utilizing

the unannotated data in the co-training strategy that would eventually lead to real improvements, I did some experiments with available annotated treebanks. The CRAFT dependency parses themselves were generated from the constituency-based treebank [1], the idea was to try to combine the CRAFT data with similarly annotated converted data. Some of them are the PTB Wall Street Journal section Marcus et al. (1993), the original GENIA treebank data Kim et al. (2003) and a version of the GENIA treebank that was already converted by the Stanford Dependency Converter. All experiments had disappointing results as any combinations between these extra data and CRAFT data lead to worse performance. Another idea was to combine CRAFT data with the Universal Dependencies (UD)'s English corpora although this also did not work.

While this could be the result of the narrow nature of the topic of the CRAFT data making it unsuitable for getting merged with other textual data, even other biomedical sources, the main reason might lie in the incompatibility of the annotations. In short, even though the CRAFT data is in the CoNLL-U format, which is supposed to be the UD version of annotation, it does not fully follow the UD annotation conventions and actually adopts some of the rules from the Stanford Dependencies (SD). Figure 5.1 illustrates some of the differences between the two types of convention.

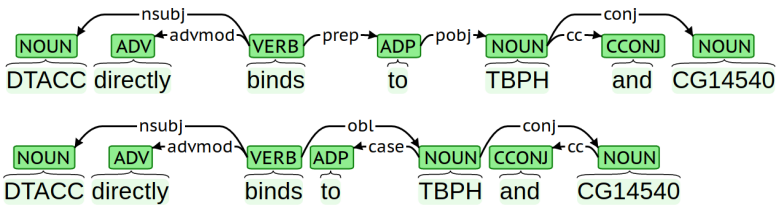

Figure 5.1: Illustration of Stanford Dependencies (top) and Universal Dependencies (bottom) analyses for an example sentence (from PMCID:15207008). The CRAFT dependency annotation arguably follows some type of a mixture between the two of them

This highly peculiar hybrid SD/UD annotation scheme significantly complicates the

[1] https://github.com/UCDenver-ccp/CRAFT/wiki/Dependency-parse-derivation-from-t

possibility of combining the CRAFT data with other annotated sources. It would be of great benefit for the bio NLP community if in the future the CRAFT data be converted fully into the UD standards (which is the *de facto* for dependency parsing now) and therefore could be added into the UD repository [2], expanding the available biomedical sources.

5.1.3 Named Entity Recognition

A named entity is any object that has a proper name, for example Helsinki is a named entity belongs to the category city, Collagen is a named entity belongs to the category protein, etc. Named Entity Recognition (NER) is an information extraction task with the goal of identifying named entities and labeling them into the correct pre-defined categories. Because the CRAFT shared tasks also included a concept annotation (CA) task [3], which means the CRAFT CA training data provided by the organizers had the correct concept annotation (or entity mention annotation), they can be incorporated into the parsing process as extra features. The segment below shows an example of a concept annotated in CRAFT. An object with the name (mention id) of *CL_basic_2014_02_21_Instance_10985* is classified into the category *spermatogonia*, a type of male germ cell

```
<annotations textSource=.17696610.txt.>
 <annotation>
  <mention id=.CL basic 2014 02 21 Instance_1098S. I>
  <annotator id=.CL_basic_2014_02_21_Instance_10000.>Mike Bada, University •
f Colorado Anschutz Medical Campus</annotator>
  <span start=.8989. end="9002" 1>
  <spannedText>spermatogonia</spannedText>
 </annotation>
 <annotation>
  <mention id=.CL basic 2014 02 21 Instance_10987. I>
  <annotator id=.CL_basic_2014_02_21_Instance_10000.>Mike Bada, University •
f Colorado Anschutz Medical Campus</annotator>
  <span start="12279" end=.12292. I>
  <spannedText>spermatogonia</spannedText>
```

The hope was that this new information would help the parser parse these named

[2] https://universaldependencies.org/

[3] https://sites.google.com/view/craft-shared-task-2019/craft-ca

entities more effectively. However the new models trained on the CRAFT data affixed with these NER categories only decreased the performance, especially on the best model without extra training data, as shown in table 5.2.

Word Embeddings	NER affixed	Dev LAS
Bio, word2vec/CBOW (window 2)	No	89.86
Bio, word2vec/CBOW (window 2)	Yes	89.66
Bio, word2vec/CBOW (window default)	No	89.55
Bio, word2vec/CBOW (window default)	Yes	89.53

Table 5.2: Development results of NER experiments

In this case, the NER tags might have become noise intruding into the learning of the parsing model.

5.2 Future Work

In this section I will take a look at the current state of applying deep learning techniques into dependency parsing in particular and natural language processing in general, as well as some predictions about the near future.

5.2.1 More transformer-based models

Since its first introduction in Vaswani et al. (2017), transformer-based models have become the most powerful and widely adopted architecture for all types of NLP tasks. In 2018, BERT, Devlin et al. (2018) an open source framework of Google, came out and achieved state-of-the-art (for its time) in many NLP tasks, and now it is still possibly the most popular transformer-based model in NLP. In 2019 we also see the releases of state-of-the-art level performance models, for example Google Brain's XLNET (Yang

et al., 2019), Facebook's RoBERTa (Liu et al., 2019b), Microsoft's MT-DNN (Liu et al., 2019a), and NVIDIA's Megatron-LM (Shoeybi et al., 2019), are all transformer-based. I expect this trend of proliferation of BERT-inspired models are not going to stop in the near future.

5.2.2 More contextual models

In recent years, the biggest breakthrough that results in many new state-of-the-art results in NLP (since the introduction of the transformer model in 2017) is the idea of *contextualized word representation* (note that the use of *contextual* here is a very specific one). In contrast to the *context-free* models such as the classic word2vec which produce only one embedding for each word (for example *book* will have the same embedding in both *I bought a book yesterday* and *I gotta book a ticket early* even if it has completely different meanings!), contextual models create a context-based word based on its sentence, thus a word might have multiple embeddings. Straka et al. (2019) extensively evaluate and compare the three methods of contexual embeddings: ELMo (Peters et al., 2018), BERT (Devlin et al., 2018) (of Google, possibly the most popular transformer-based model in NLP now), Flair (Akbik et al., 2019) in three familiar tasks: part-of-speech tagging, lemmatization, and dependency parsing. Using the system of UDpipe, all three of them substantially improved upon all three tasks in the CoNLL 2018 shared tasks.

I expect to see more contextual models coming in 2020. In the words of Jeff Dean, the AI Chef of Google , "We'd still like to be able to do much more contextual kinds of models. Like right now BERT and other models work well on hundreds of words, but not 10,000 words as context. So that's kind of [an] interesting direction." [4]

[4] https://venturebeat.com/2020/01/02/top-minds-in-machine-learning-predict-wher

5.2.3 The future of dependency parsing

As mentioned in section 2.2, dependency parsing has a long history, and is a very important component in many NLP-related tasks with most NLP toolkits available having the ability to do dependency parsing. However parsing is usually just a constituent (or building block) that can help enhance the performance of downstream applications, for example in question answering, in machine translation and many more.

In the era of "going deeper" with the NLP scene completely dominated by deep neural end-to-end models, dependency parsing can at best, be utilized as providing extra useful information as a good parsing model will have been trained on a very large set of textual data, and its learning goal, the grammar analysis, is fairly universal and easily generalized across languages and tasks, or at worst, can be done without. Nevertheless, I still have a hopeful vision about the future of parsing as a look at the number of academic papers published mentioning dependency parsing since 2019 up until the very beginning of 2020 indicate, there are still many people out there doing research on this NLP task. I hope to see more breakthroughs related to dependency parsing in the near future.

6 Conclusions

In this thesis, I have presented my work as a member of the TurkuNLP team taking part in the project of the CRAFT 2019 shared task in structural analysis. The majority of the experiments was performed on the Turku neural parser, the main parsing system, with some consideration to UDpipe. The Turku parser with its four components tokenizer, tagger, lemmatizer and tagger, all have deep learning based architecture, provided a strong baseline parsing performance already on the out-of-domain CoNLL English word embeddings, without any modification on the training data. A variety of strategies were adopted to improve on this, many of them did not prove to be fruitful, nevertheless a few stood out as particularly effective.

In-domain biomedical word embeddings can enhance the parsing model, providing a great set of initiated word representations as one of the inputs of the parser. While some previously pre-trained word embeddings failed to provide substantial improvement, newly made custom word vectors trained on large sources of unannotated biomedical data from PMC and Pubmed proved very effective.

A modified version of the co-training technique pushed the parser even further, using the same sources of unannotated data. After multiple carefully supervised experiments, the amount of extra training data needed reached its sweet spot at around 4 thousand sentences from PMC full-text articles.

The incorporation of these two strategies into the already strong Turku parser helped the TurkuNLP team achieve the highest results on the test set of the CRAFT shared task

with a LAS score of 89.695%.

A few recommendations on future work were also discussed. A fully-converted into universal dependencies standard version of the CRAFT data is in demand, as that would prove invaluable for the bio NLP community. To continue improving upon the model accuracy in terms of dependency parsing, contextual word representations should be taken into account, as well as the employment of deeper transformer-based architectures. And finally, I hope that dependency parsing itself would be utilized more in other NLP tasks, such as machine translation and chat bots.

References

Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, Vancouver, Canada, Aug. 2017. Association for Computational Linguistics. doi: 10.18653/v1/K17-3. URL https://www.aclweb.org/anthology/K17-3000.

Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin. *Learning From Data*. AMLBook, 2012. ISBN 1600490069, 9781600490064.

S. Agarwal and H. Yu. Detecting hedge cues and their scope in biomedical text with conditional random fields. *Journal of Biomedical Informatics*, 43(6):953 – 961, 2010. ISSN 1532-0464. doi: https://doi.org/10.1016/j.jbi.2010.08.003. URL http://www.sciencedirect.com/science/article/pii/S1532046410001140.

A. Airola, S. Pyysalo, J. Björne, T. Pahikkala, F. Ginter, and T. Salakoski. All-paths graph kernel for protein-protein interaction extraction with evaluation of cross-corpus learning. *BMC Bioinformatics*, 9:S2 – S2, 2008.

A. Akbik, T. Bergmann, D. Blythe, K. Rasul, S. Schweter, and R. Vollgraf. FLAIR: An easy-to-use framework for state-of-the-art NLP. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-4010. URL https://www.aclweb.org/anthology/N19-4010.

A. N. Ananthakrishnan, T. Cai, G. Savova, S.-C. Cheng, P. Chen, R. G. Perez, V. S. Gainer, S. N. Murphy, P. Szolovits, Z. Xia, S. Shaw, S. Churchill, E. W. Karlson, I. Kohane, R. M. Plenge, and K. P. Liao. Improving Case Definition of Crohn's Disease and Ulcerative Colitis in Electronic Medical Records Using Natural Language Processing: A Novel Informatics Approach. *Inflammatory Bowel Diseases*, 19(7):1411–1420, 04 2013. ISSN 1078-0998. doi: 10.1097/MIB.0b013e31828133fd. URL `https://doi.org/10.1097/MIB.0b013e31828133fd`.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

W. Bao, J. Yue, and Y. Rao. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PLoS ONE*, 12, 07 2017. doi: 10.1371/journal.pone.0180944.

J. Björne, J. Heimonen, F. Ginter, A. Airola, T. Pahikkala, and T. Salakoski. Extracting contextualized complex biological events with rich graph-based feature sets. *Computational Intelligence*, 27:541–557, 11 2011. doi: 10.1111/j.1467-8640.2011.00399.x.

L. Bloomfield and C. Hockett. *Language*. University of Chicago Press, 1933. ISBN 9780226060675. URL `https://books.google.fi/books?id=87BCDVsmFE4C`.

P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

S. Buchholz and E. 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, New York City, June 2006. Association for Computational Linguistics.

K. Cao and M. Rei. A joint model for word embedding and word morphology. In *Rep4NLP@ACL*, 2016.

E. Charniak and M. Johnson. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 173–180, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics. doi: 10.3115/1219840.1219862. URL https://www.aclweb.org/anthology/P05-1022.

D. Chen and C. 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, Doha, Qatar, Oct. 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1082. URL https://www.aclweb.org/anthology/D14-1082.

B. Chiu, G. Crichton, A. Korhonen, and S. Pyysalo. How to train good word embeddings for biomedical nlp. pages 166–174, 01 2016. doi: 10.18653/v1/W16-2922.

K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014a. URL http://arxiv.org/abs/1406.1078.

K. Cho, B. van Merriënboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. 06 2014b. doi: 10.3115/v1/D14-1179.

J. D. Choi and M. Palmer. Guidelines for the clear style constituent to dependency conversion. *Technical Report 01–12*, 2012.

N. Chomsky. *Syntactic Structures*. Mouton and Co., The Hague, 1957.

R. Cotterell, C. Kirov, J. Sylak-Glassman, G. Walther, E. Vylomova, P. Xia, M. Faruqui, S. Kübler, D. Yarowsky, J. Eisner, and M. Hulden. Conll-sigmorphon 2017 shared task: Universal morphological reinflection in 52 languages. *CoRR*, abs/1706.09031, 2017. URL http://arxiv.org/abs/1706.09031.

K. Crammer and Y. Singer. Ultraconservative online algorithms for multiclass problems. *J. Mach. Learn. Res.*, 3:951–991, Mar. 2003. ISSN 1532-4435. doi: 10.1162/jmlr.2003.3.4-5.951. URL https://doi.org/10.1162/jmlr.2003.3.4-5.951.

M.-C. de Marneffe and C. D. Manning. The stanford typed dependencies representation. In *Coling 2008: Proceedings of the Workshop on Cross-Framework and Cross-Domain Parser Evaluation*, CrossParser '08, pages 1–8, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics. ISBN 978-1-905593-50-7. URL http://dl.acm.org/citation.cfm?id=1608858.1608859.

J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

T. Dozat and C. D. Manning. Deep biaffine attention for neural dependency parsing. *CoRR*, abs/1611.01734, 2016. URL http://arxiv.org/abs/1611.01734.

T. Dozat, P. Qi, and C. D. Manning. Stanford's graph-based neural dependency parser at the CoNLL 2017 shared task. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 20–30, Vancouver, Canada, Aug. 2017. Association for Computational Linguistics. doi: 10.18653/v1/K17-3002. URL https://www.aclweb.org/anthology/K17-3002.

J. M. Eisner. Three new probabilistic models for dependency parsing: An exploration. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*, 1996. URL https://www.aclweb.org/anthology/C96-1058.

J. R. Firth. A synopsis of linguistic theory 1930-55. 1952-59:1–32, 1957.

A. Galassi, M. Lippi, and P. Torroni. Attention, please! A critical review of neural attention models in natural language processing. *CoRR*, abs/1902.02181, 2019. URL http://arxiv.org/abs/1902.02181.

M. Galley and C. D. Manning. Quadratic-time dependency parsing for machine translation. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2*, ACL '09, page 773–781, USA, 2009. Association for Computational Linguistics. ISBN 9781932432466.

F. Ginter, J. Hajič, J. Luotolahti, M. Straka, and D. Zeman. CoNLL 2017 shared task - automatically annotated raw texts and word embeddings, 2017. URL http://hdl.handle.net/11234/1-1989. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Z. Harris. Distributional structure. *Word*, 10(23):146–162, 1954.

D. G. Hays. Automatic language-data processing. 1962.

J. Henderson. Discriminative training of a neural network statistical parser. In *Proceedings of the 42Nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics. doi: 10.3115/1218955.1218968. URL https://doi.org/10.3115/1218955.1218968.

S. Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München, 1991.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9: 1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79(8):2554–2558, Apr. 1982. ISSN 0027-8424. URL http://view.ncbi.nlm.nih.gov/pubmed/6953413].

G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014. ISBN 1461471370, 9781461471370.

A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. Bag of tricks for efficient text classification. *CoRR*, abs/1607.01759, 2016. URL http://arxiv.org/abs/1607.01759.

J. Kanerva, J. Luotolahti, and F. Ginter. TurkuNLP: Delexicalized pre-training of word embeddings for dependency parsing. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 119–125, Vancouver, Canada, Aug. 2017. Association for Computational Linguistics. doi: 10.18653/v1/K17-3012. URL https://www.aclweb.org/anthology/K17-3012.

J. Kanerva, F. Ginter, N. Miekka, A. Leino, and T. Salakoski. Turku neural parser pipeline: An end-to-end system for the conll 2018 shared task. In *CoNLL Shared Task*, 2018.

J. Kanerva, F. Ginter, and T. Salakoski. Universal lemmatizer: A sequence to sequence model for lemmatizing universal dependencies treebanks. *CoRR*, abs/1902.00972, 2019. URL http://arxiv.org/abs/1902.00972.

P. Kanerva, J. Kristoferson, and A. Holst. Random indexing of text samples for latent semantic analysis. 2000.

F. Karlsson, A. Voutilainen, J. Heikkilä, and A. Anttila. *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*. 01 1995.

J.-D. Kim, T. Ohta, Y. Tateisi, and J. Tsujii. GENIA corpus—a semantically annotated corpus for bio-textmining. *Bioinformatics*, 19(suppl$_1$) : $i180--i182, 072003. ISSN1367-4803. doi$: . URL https://doi.org/10.1093/bioinformatics/btg1023.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

E. Kiperwasser and Y. Goldberg. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327, 2016a. 10.1162/tacl$_{a0}$0101.URL.

E. Kiperwasser and Y. Goldberg. Easy-first dependency parsing with hierarchical tree lstms. *CoRR*, abs/1603.00375, 2016b. URL http://arxiv.org/abs/1603.00375.

G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush. Opennmt: Open-source toolkit for neural machine translation. *CoRR*, abs/1701.02810, 2017. URL http://arxiv.org/abs/1701.02810.

D. E. Knuth. On the translation of languages from left to right. *Information and Control*, 8 (6):607 – 639, 1965. ISSN 0019-9958. https://doi.org/10.1016/S0019-9958(65)90426-2. URL http://www.sciencedirect.com/science/article/pii/S00199958659042

A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012. 10.1145/3065386.

S. Kulick, A. Bies, M. Liberman, M. Mandel, R. Mcdonald, M. Palmer, A. Schein, L. Ungar, S. Winters, and P. White. Integrated annotation for biomedical information extraction. *Proceedings of the HLT-NAACL-04 Workshop on Linking Biological Literature, Ontologies and Databases, Boston, MA*, 01 2004.

H. Liu, L. Hunter, V. Keselj, and K. Verspoor. Approximate subgraph matching-based literature mining for biomedical events and relations. *PloS one*, 8:e60954, 04 2013. 10.1371/journal.pone.0060954.

X. Liu, P. He, W. Chen, and J. Gao. Multi-task deep neural networks for natural language understanding. *CoRR*, abs/1901.11504, 2019a. URL http://arxiv.org/abs/1901.11504.

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019b. URL http://arxiv.org/abs/1907.11692.

Y. Luo, O. Uzuner, and P. Szolovits. Bridging semantics and syntax with graph algorithms-state-of-the-art of extracting biomedical relations. *Briefings in bioinformatics*, 18, 02 2016. 10.1093/bib/bbw001.

Y. Luo, Ö. Uzuner, and P. Szolovits. Bridging semantics and syntax with graph algorithms - state-of-the-art of extracting biomedical relations. *Briefings in bioinformatics*, 18 1: 160–178, 2017.

M. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015. URL http://arxiv.org/abs/1508.04025.

C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, Baltimore, Maryland, June 2014. Association for Computational Linguistics. 10.3115/v1/P14-5010. URL https://www.aclweb.org/anthology/P14-5010.

M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993. URL https://www.aclweb.org/anthology/J93-2004.

D. McClosky and E. Charniak. Self-training for biomedical parsing. In *Proceedings of ACL-08: HLT, Short Papers*, pages 101–104, Columbus, Ohio, June 2008. Association

for Computational Linguistics. URL https://www.aclweb.org/anthology/P08-2026.

W. S. McCulloch and W. Pitts. A logical calculus of the idea immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

R. McDonald, K. Crammer, and F. Pereira. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 91–98, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics. 10.3115/1219840.1219852. URL https://doi.org/10.3115/1219840.1219852.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, 2013, 01 2013.

M. Miwa, S. Pyysalo, T. Hara, and J. Tsujii. A comparative study of syntactic parsers for event extraction. pages 37–45, 01 2010.

Y. Miyao, R. Sætre, K. Sagae, and J. Tsujii. Task-oriented evaluation of syntactic parsers and their representations. pages 46–54, 01 2008.

D. Q. Nguyen, M. Dras, and M. Johnson. A novel neural network model for joint POS tagging and graph-based dependency parsing. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 134–142, Vancouver, Canada, Aug. 2017. Association for Computational Linguistics. 10.18653/v1/K17-3014. URL https://www.aclweb.org/anthology/K17-3014.

J. Nivre and M. Scholz. Deterministic dependency parsing of English text. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 64–70, Geneva, Switzerland, aug 23–aug 27 2004. COLING. URL https://www.aclweb.org/anthology/C04-1010.

J. Nivre, J. Hall, and J. Nilsson. Maltparser: A data-driven parser-generator for dependency parsing. 01 2006.

Pāini and S. Katre. *Aādhyāyī of Pāini*. Texas Pan American Series. University of Texas Press, 1987. ISBN 9780292703940. URL https://books.google.fi/books?id=0BKwAAAAIAAJ.

N. Peng, H. Poon, C. Quirk, K. Toutanova, and W. Yih. Cross-sentence n-ary relation extraction with graph lstms. *CoRR*, abs/1708.03743, 2017. URL http://arxiv.org/abs/1708.03743.

P. Percival. Fitch and intuitionistic knowability. *Analysis*, 50(3):182–187, 06 1990. ISSN 0003-2638. 10.1093/analys/50.3.182. URL https://doi.org/10.1093/analys/50.3.182.

M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018. URL http://arxiv.org/abs/1802.05365.

S. Petrov, D. Das, and R. McDonald. A universal part-of-speech tagset. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 2089–2096, Istanbul, Turkey, May 2012. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2012/pdf/274_Paper.pdf.

S. Pyysalo, F. Ginter, K. Haverinen, J. Heimonen, T. Salakoski, and V. Laippala. On the unification of syntactic annotations under the stanford dependency scheme: A case study on bioinfer and genia. In *Proceedings of the Workshop on BioNLP 2007: Biological, Translational, and Clinical Language Processing*, BioNLP '07, pages 25–32, Stroudsburg, PA, USA, 2007a. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=1572392.1572397.

S. Pyysalo, F. Ginter, J. Heimonen, J. Björne, J. Boberg, J. Järvinen, and T. Salakoski. Bioinfer: A corpus for information extraction in the biomedical domain. *BMC bioinformatics*, 8:50, 02 2007b. 10.1186/1471-2105-8-50.

S. Pyysalo, F. Ginter, H. Moen, T. Salakoski, and S. Ananiadou. Distributional semantics resources for biomedical text processing. 2013.

B. D. Ripley and N. L. Hjort. *Pattern Recognition and Neural Networks*. Cambridge University Press, New York, NY, USA, 1st edition, 1995. ISBN 0521460867.

K. Sagae and J. Tsujii. Dependency parsing and domain adaptation with LR models and parser ensembles. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 1044–1050, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D07-1111.

G. Schneider, F. Rinaldi, and J. Dowdall. Fast, deep-linguistic statistical minimalist dependency parsing. 01 2004.

M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Trans. Signal Processing*, 45:2673–2681, 1997.

M. Shoeybi, M. M. A. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro. Megatron-lm: Training multi-billion parameter language models using gpu model parallelism. 09 2019.

D. D. Sleator and D. Temperley. Parsing english with a link grammar. *CoRR*, abs/cmp-lg/9508004, 1995. URL http://arxiv.org/abs/cmp-lg/9508004.

M. Straka and J. Straková. Tokenizing, POS tagging, lemmatizing and parsing UD 2.0 with UDPipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Vancouver, Canada, Aug. 2017. Association for Computational Linguistics. 10.18653/v1/K17-3009. URL https://www.aclweb.org/anthology/K17-3009.

M. Straka, J. Straková, and J. Hajic. Evaluating contextualized embeddings on 54 languages in pos tagging, lemmatization and dependency parsing. *ArXiv*, abs/1908.07448, 2019.

H. Sugiyama, T. Meguro, R. Higashinaka, and Y. Minami. Open-domain utterance generation for conversational dialogue systems using web-scale dependency structures. In *Proceedings of the SIGDIAL 2013 Conference*, pages 334–338, 2013.

M. Sundermeyer, R. Schlüter, and H. Ney. Lstm neural networks for language modeling. 09 2012.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014. URL `http://arxiv.org/abs/1409.3215`.

L. Tesnière, T. Osborne, and S. Kahane. *Elements of Structural Syntax*. OAPEN Library. John Benjamins Publishing Company, 1959. ISBN 9789027212122. URL `https://books.google.fi/books?id=FNjooAEACAAJ`.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL `http://arxiv.org/abs/1706.03762`.

K. Verspoor, K. Cohen, A. Lanfranchi, C. Warner, H. Johnson, and C. Roeder. A corpus of full-text journal articles is a robust evaluation tool for revealing differences in performance of biomedical natural language processing tools. *BMC Bioinform*, 13:1–26, 01 2012a.

K. Verspoor, K. B. Cohen, A. Lanfranchi, C. Warner, H. L. Johnson, C. Roeder, J. D. Choi, C. Funk, Y. Malenkiy, M. Eckert, et al. A corpus of full-text journal articles is a robust evaluation tool for revealing differences in performance of biomedical natural language processing tools. *BMC bioinformatics*, 13(1):207, 2012b.

S. Vieira, W. Pinaya, and A. Mechelli. Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications. *Neuroscience Biobehavioral Reviews*, 74, 01 2017. 10.1016/j.neubiorev.2017.01.002.

Y. Wang, S. Pakhomov, J. O. Ryan, and G. B. Melton. Domain adaption of parsing for operative notes. *Journal of Biomedical Informatics*, 54:1 – 9, 2015. ISSN 1532-0464.

https://doi.org/10.1016/j.jbi.2015.01.016. URL http://www.sciencedirect.com/science/article/pii/S1532046415000180.

D. Weiss, C. Alberti, M. Collins, and S. Petrov. Structured training for neural network transition-based parsing. *CoRR*, abs/1506.06158, 2015. URL http://arxiv.org/abs/1506.06158.

H. Yamada and Y. Matsumoto. Statistical dependency analysis with support vector machines. 2003.

Z. Yang, Z. Dai, Y. Yang, J. G. Carbonell, R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, abs/1906.08237, 2019. URL http://arxiv.org/abs/1906.08237.

D. Zeman. Reusable tagset conversion using tagset drivers. 01 2008.

D. Zeman, M. Popel, M. Straka, J. Hajič, J. Nivre, F. Ginter, J. Luotolahti, S. Pyysalo, S. Petrov, M. Potthast, et al. Conll 2017 shared task. *Proceedings of the CoNLL 2017 Shared Task Multilingual Parsing from Raw Text to Universal Dependencies*, 2017.

D. Zeman, J. Hajič, M. Popel, M. Potthast, M. Straka, F. Ginter, J. Nivre, and S. Petrov. Conll 2018 shared task: multilingual parsing from raw text to universal dependencies. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 1–21, 2018.

Y. Zhang, F. Tiryaki, M. Jiang, and H. Xu. Parsing clinical text using the state-of-the-art deep learning based parsers: a systematic comparison. In *BMC Medical Informatics and Decision Making*, 2019.

R. Zhao, D. Wang, R. Yan, K. Mao, F. Shen, and J. Wang. Machine health monitoring using local feature-based gated recurrent unit networks. *IEEE Transactions on Industrial Electronics*, 65(2):1539–1548, Feb 2018. 10.1109/TIE.2017.2733438.