Dependency Parsing and Dialogue Systems

an investigation of dependency parsing for commercial application

Allison Adams

Uppsala University
Department of Linguistics and Philology
Master’s Programme in Language Technology
Master’s Thesis in Language Technology
June 19, 2017

Supervisors:
Miryam de Lhoneux, Uppsala University
Rebecca Jonsson, Artificial Solutions Iberia SL
Abstract

In this thesis, we investigate dependency parsing for commercial application, namely for future integration in a dialogue system. To do this, we conduct several experiments on dialogue data to assess parser performance on this domain, and to improve this performance over a baseline. This work makes the following contributions: first, the creation and manual annotation of a gold-standard data set for dialogue data; second, a thorough error analysis of the data set, comparing neural network parsing to traditional parsing methods on this domain; and finally, various domain adaptation experiments show how parsing on this data set can be improved over a baseline. We further show that dialogue data is characterized by questions in particular, and suggest a method for improving overall parsing on these constructions.
Contents

Acknowledgements 5

1 Introduction 6
  1.1 Purpose and scope .................................. 6
  1.2 Outline of thesis .................................. 7

2 Background 8
  2.1 Dependency syntax ................................ 8
    2.1.1 Dependency grammar ........................... 8
  2.2 Data-driven approaches to dependency parsing .... 10
    2.2.1 Dependency parsing ............................. 10
    2.2.2 Possible advantages of dependency syntax in NLP . 11
    2.2.3 Transition-based parsing ....................... 12
    2.2.4 History-based feature models ................. 13
    2.2.5 Parsing with neural networks .................. 14
    2.2.6 Evaluation metrics ............................ 17
  2.3 Annotation schemes for dependencies .......... 18
  2.4 Error analyses of dependency parsers .......... 19
  2.5 Domain adaptation for parsing .................. 22
  2.6 Dialogue systems and conversational agents .... 23

3 The Dataset 26
  3.1 Annotating the dataset ......................... 26
  3.2 Analyzing the dataset ........................... 28
  3.3 Categorizing the data set ...................... 31
  3.4 Discussion ..................................... 33

4 Parsing the data 34
  4.1 Overall model accuracy .......................... 34
  4.2 Detailed accuracy analysis ...................... 35
  4.3 Discussion ..................................... 39

5 Adapting the parser model 40
  5.1 Building a better treebank ..................... 40
    5.1.1 Co-training .................................. 40
    5.1.2 Tri-training ................................. 40
    5.1.3 Using QuestionBank .......................... 41
    5.1.4 Results .................................... 41
  5.2 Selecting for sentence length .................. 42
5.3 Question handling ........................................... 42
5.4 Discussion .................................................. 43

6 Conclusion and future work 45
6.1 Using dependency-based syntactic annotation in practice .... 45
6.2 Conclusion .................................................. 45

Bibliography 47
Acknowledgements

I would like to thank my advisor, Miryam de Lhoneux, for helping to keep this work focused through her constant reminders that a thesis ought to follow a cohesive narrative. I would also like to thank my supervisor, Rebecca Jonsson, for giving me the opportunity to work as a part of the Artificial Solutions research team for the duration of this project. Finally, I would like to express my immense gratitude to both the faculty and my fellow students of the language technology program in Uppsala for their invaluable support and guidance these past two years.
1 Introduction

1.1 Purpose and scope

In this thesis, we investigate dependency parsing on a narrow and specific domain of English: dialogue data. In recent years, dialogue systems have become increasingly prevalent in day-to-day life—it is not uncommon to interact with a dialogue system when booking a flight, making a hands-free phone call while driving, or even while online shopping. The conversations that result from these interactions often differ from a typical conversation with a human being, and as a result, deriving their underlying syntactic structures can be difficult for human experts and automatic parsers alike. As such, integrating syntactic information in a dialogue system poses a particular challenge. This thesis addresses these challenges in the following ways:

- By annotating dialogue test data, and conducting a linguistic and statistical analysis of that data so as to quantify the differences between this domain and standard English treebank data.
- By analyzing dependency parser performance on dialogue data through an in-depth error analysis, and by investigating the relative parser performance of traditional and neural network-based models on our data set.
- By investigating methods to adapt parsers to the specificities of dialogue data through the automatic annotation of in-domain treebanks, as well as preprocessing of the input data.

We base our annotation strategy on the guidelines set by the Universal Dependencies project (Nivre et al., 2016), to allow us to compare our data set to other English treebanks. For our error analysis, we follow the techniques described in McDonald and Nivre (2007), in which the authors characterized dependency parser errors by examining both linguistic and graph-based factors. Our domain adaptation experiments are based on research showing the value of automatically creating in-domain data for parser training (Baucom et al., 2013; Sagae and Tsujii). So far, many parsing studies on English data have concerned standard English treebanks, with little focus on dependency parsing for application in a commercial setting. To the best of our knowledge, this work constitutes the first linguistic characterization and dependency parser error analysis of English dialogue system data. We further contribute domain adaptation strategies based on mitigating challenges specific to this domain. This data for this study were generously provided by Artificial Solutions, a language technology company that, among other services, provides a platform for the streamlined production of customer-service dialogue systems. As the company grows and takes on different kinds of clients, they have begun to look at ways to improve upon their
existing platform, including the integration of certain NLP tools. As such, a goal of this project is to test parsing on dialogue data as a first step toward an integration of a dependency parser in a dialogue system. All of our data were taken from customer conversations with these dialogue systems.

1.2 Outline of thesis

Chapter 2 provides an overview of prior work relevant to this thesis. We introduce the concepts of dependency grammar and data-driven dependency parsing, and describe how recent advances in the field compare to traditional methods. We give a description of different dependency annotation schemes, and we further discuss error analyses of dependency parsers, as well as provide an overview of domain adaptation for dependency parsing. Finally, we present the general concepts relating to dialogue systems and conversational agents.

Chapter 3 presents our process of annotating conversational data, followed by a detailed linguistic analysis of the data. We discuss the characteristics that define the domain of dialogue data and the challenges posed by this domain to both annotation and parsing.

Chapter 4 describes our parsing experiments on a dialogue data set and compares the relative success of newer neural network parsers to a traditional dependency parser. We carry out an in-depth error analysis on our parsers’ output to compare the parser models, and to quantify what aspects of our domain are particularly difficult for parsing.

Chapter 5 discusses the results of our domain adaptation experiments on dialogue data, in which we attempt to adapt a parser to mitigate errors on the dialogue data set. We discuss several different methods that we tested, which involved both creating and altering treebank data, as well as pre-processing input data. We also discuss how parsing can be improved for questions, which make up a significant portion of our dialogue test set.

Chapter 6 summarizes our findings and makes suggestions for future research. In particular, we discuss potential areas of application of dependency-based syntactic information in dialogue systems.
2 Background

2.1 Dependency syntax

2.1.1 Dependency grammar

Dependency syntax, though somewhat overlooked by 20th century linguists, has been gaining popularity among NLP researchers in recent decades. Modern theories of dependency grammar are often attributed to the work of Lucien Tesnière, who described his dependency grammar as follows (Nivre, 2005):

The sentence is an organized whole, the constituent elements of which are words. Every word that belongs to a sentence ceases by itself to be isolated as in the dictionary. Between the word and its neighbors, the mind perceives connections, the totality of which forms the structure of the sentence. The structural connections establish dependency relations between the words. Each connection in principle unites a superior term and an inferior term. The superior term receives the name governor. The inferior term receives the name subordinate. Thus, in the sentence Alfred parle [...], parle is the governor and Alfred the subordinate.

Tesnière’s theory was based upon the notion that the words in a sentence are connected to each other by asymmetric directed links called dependency relations (or, simply, dependencies). A dependency relation consists of a governing node, often referred to as the head and its dependent, which is considered to be syntactically subordinate to the head (Kübler et al., 2009). This way of understanding the underlying syntactic relations in a sentence is distinct from the 20th century’s dominating syntactic paradigm, phrase structure grammar (Chomsky, 1957). Figure 2.1 provides an example of a phrase structure grammar representation of a sentence. As the name implies, in a phrase structure representation, the sentence is grouped hierarchically into phrases, with each phrase classified according its structural category. For example, the phrase Economic news is grouped together and classified as a NP or Noun Phrase, and that NP in turn is grouped together with a VP (Verb Phrase) and PU (Punctuation) to be governed by the phrasal node S (Sentence).

Figure 2.2, which depicts a dependency graph for the same sentence, exemplifies how the two systems differ. In this structure, relations are made between words instead of phrases, with these relations classified by functional categories, like subject (nsubj) and object (dobj). In this representation, each arrow forms a directed arc between the words in the sentence, representing the dependency relation between each word. Each arc edge features a dependency label, which describes the type of dependency between the head and the dependent. Here, an artificial ROOT node is inserted at the beginning of the sentence, and serves as the head to the sentence’s main predicate.
verb. This ROOT node acts as a sort of placeholder, which, among other things, serves to simplify computational implementation (Kabler et al., 2009).

In dependency grammar, as with constituency-based frameworks, it is necessary to establish criteria for building dependency relations, namely, to identify which word is the head and which is the dependent in these relations. Nivre (2005) lists some of the various criteria suggested by Zwicky (1985) and Hudson (1984) for identifying a syntactic relations between a head H and dependent D in a construction C:

1. H determines the syntactic category of C and can often replace C.
2. H determines the semantic category of C; D gives semantic specification.
3. H is obligatory; D may be optional.
4. H selects D and determines whether D is obligatory or optional.
5. The form of D depends on H (agreement or government).
6. The linear position of D is specified with reference to H.

This list contains a mixture of both syntactic and semantic criteria, however, raising the question as to how these criteria may be combined to fit one coherent definition of dependency. To address this coherence problem, some theorists have suggested that these criteria may apply to different layers of the dependency structure, a syntactic, semantic, and morphological layer, or a surface structure and a deep structure. Other theorists still have suggested a need for different kinds of criteria for different types of constructions, particularly endocentric and exocentric constructions.
(Kübler et al., 2009). For example, in the dependency graph in Figure 2.2, the amod relation between financial and markets is endocentric, that is to say, the dependent can be removed without disruption to the syntactic structure. This can not be said for other types of dependency relations, where the relation is exocentric, and cannot be replaced by the head without disrupting the syntactic structure (for example the subject and object relations). These types of relations necessarily fail the first criterion.

Other types of relations are difficult to classify as either endo- or exocentric, for example the relation of on to markets. This distinction (or in some case difficulty to distinguish) between these types of syntactic structures is closely related the concept of head-complement and head-modifier relations featured in modern theories of syntax. While these types of structures are, in most cases, very clearly defined in dependency grammar, there are also some constructions that are seemingly unaccounted for by dependency syntax. These include cases where the direction of the dependency is unclear, as is often the case for grammatical function words (articles, auxiliaries, prepositions). For these types of constructions, traditional theories of dependency have not provided a concrete solution as to what should be regarded as the head and what should be treated as the dependent, or even if these constructions constitute dependencies at all (Kübler et al., 2009). This is similarly the case for coordination, which puts forth a scenario in which two different heads are equally plausible. For these types of constructions, while certain interpretations may be more popular, there is no clear consensus.

2.2 Data-driven approaches to dependency parsing

2.2.1 Dependency parsing

According to Kübler et al. (2009) a dependency graph can be defined as follows. Firstly, a sentence is defined as a sequence of tokens. This sequence is assumed to be tokenized, as dependency parsers operate exclusively on pre-tokenized input. As was mentioned in a prior section and depicted in Figure 2.2, an artificial root node is inserted in the first position in the sentence. Each token represents a word (word here is used liberally, as the term token may also be used to refer to punctuation markers or morphemes). Secondly, Kübler et al. (2009) assume that a finite set of possible dependency labels types, all of which can hold between any two words in a sentence, exists. Following these two definitions, a dependency graph can be defined as a labeled directed graph consisting of nodes and arcs, in which tokens correspond directly to the nodes, and the dependency type labels the arcs between them. In other words, a dependency graph is a set of dependency relations between the words of a sentence. A well formed dependency graph, also called a dependency tree, is a dependency graph that originates from the root node, and spans the entirety of a sentence. Moreover, Kübler et al. (2009) note that the following properties must also apply to a dependency tree:

- **Connectedness**: when direction is ignored, there exists a path in the dependency tree connecting every two words. This is equivalent to graph theory’s notion of a weakly connected directed graph. Basically, every node in the dependency graph must be connected to at least one other node.
• **Single-head:** each node in the dependency tree cannot have more than one head. Generally, the only node without a head is the artificial root node.

• **Acyclicity:** a dependency tree cannot contain any cycles as in such a scenario words would be dependent upon themselves.

There are also constraints placed on dependency trees that are not inherent in the definition of dependency graphs, but rather are restrictions put in place by computational systems, the most common of these being limiting parsing to projective dependency trees. A non-projective dependency tree occurs when a dependent is sequentially separated from the head in a sentence, resulting in two dependency arcs intersecting one another. Consider, for example, Figure 2.3:

![Non-projective dependency tree](image)

**Figure 2.3:** Non-projective dependency tree for an English sentence

As the dependency tree seen here can only be drawn with edges crossing one another, this is a non-projective dependency tree. Because projective dependency trees satisfy the *nested property*, that is to say, their dependency relations are nested and do not intersect one another, many computational dependency parsing systems enforce projectivity, as this lends some computational advantages. In English in particular, which contains a very low percentage of non-projective structures, a system’s inability to handle non-projective trees tends to have little to no effect on its overall accuracy.

### 2.2.2 Possible advantages of dependency syntax in NLP

Dependency grammar differs from traditional constituency grammars most fundamentally in its lack of phrasal nodes, with proponents of dependency parsing citing several advantages over constituency parsing. According to [Covington (2001)](covington), the advantages of dependency parsing can be described as follows:

• Dependency links are close to the semantic relationships needed for the next stage of interpretation; it is not necessary to “read off” head-modifier or head-complement relations from a tree that does not show them directly.

• The dependency tree contains one node per word. Because the parser’s job is only to connect existing nodes, not to postulate new ones, the task of parsing is in some sense more straightforward. [...] 

• Dependency parsing lends itself to word-at-a-time operation, i.e., parsing by accepting and attaching words one at a time rather than by waiting for complete phrases. [...] 

Additionally, [Nivre (2005)](nivre) notes that while dependency parsing is in some ways a less expressive syntactic representation, he posits that, because greater expressivity
corresponds to more complex syntactic parsing, dependency representations may, in fact, strike an optimal balance in this respect. The lack in expressivity, Nivre argues, is compensated for by means of a relatively directly encoded predicate-argument structure and by a bilexical relational composition, both of which are useful for deriving a semantic analysis of the sentence. Due to these factors, dependency structures are expressive enough to be useful to NLP systems, but simple enough so as to allow for full parsing with high accuracy and efficiency (Nivre, 2005).

2.2.3 Transition-based parsing

Dependency parsing is the process by which dependency relations are assigned between the words in a sentence. Kübler et al. (2009) divide dependency parsing approaches into two classes: grammar-based dependency parsing and data-driven dependency parsing, with grammar-based approaches relying on a formal grammar, and data-driven approaches using machine learning from linguistic data. All parsing systems used in the course of this thesis use data-driven methods. Data-driven dependency parsing can be broken down in to two major problems: learning, in which training data is used to induce a parsing model, and parsing, in which the learned parsing model is used to derive the dependency graphs of new, unseen sentences. The learning and parsing tasks are solved in different ways by different classes of parser, with one popular class of parsers being transition-based parsers.

A transition-based dependency parser works by starting at an initial configuration, and, using a guide, choosing between one of several transitions (actions). The parser stops once it reaches a target terminal configuration, and returns the dependency tree associated with that configuration. A configuration for a sentence comprises three structures: a buffer, a stack, and a dependency graph (a set of arcs). In the initial configuration, all the words in a sentence are in the buffer, and the stack contains only the artificial root node, and the set of arcs is empty. The algorithm terminates when the buffer is empty (Kübler et al., 2009). The basic idea is that as the parser processes the sentence from left to right, at each transition at each token, the configuration represents a partially parsed sentence, where the words on the stack are partially processed, those in the buffer are the words in the input sentence that have yet to be processed, and the set of arcs contains the completed parts of the dependency tree (Kübler et al., 2009). Each transition either adds an arc to the dependency tree, or modifies the stack or buffer. In shift-reduce parsing, the possible transitions are:

- LEFT-ARC: for a dependency label l adds an arc (j,l,i), where i is the top node of the stack, and j is the first node in the buffer. This action also pops the stack, and has a precondition that both the stack and the buffer are non-empty, and is not the root node.
- RIGHT-ARC: for a dependency label l adds a dependency arc (i,l,j), where i is the top node of the stack, and j is the first node in the buffer. This action pops the stack, and additionally replaces j with i at the head of the buffer. This action must also satisfy the condition that both the stack and the buffer are non-empty.
- SHIFT: removes first node in the buffer, and pushes it to the top of the stack. It has a precondition that the buffer is non-empty

Each transition represents an elementary parsing action, while a set of transitions, or a transition sequence, results in a fully-formed dependency tree. The transition
system in itself, however, is non-deterministic, as at each configuration multiple transitions may be permissible. In order to derive a full transition sequence, a transition system must be supplemented a mechanism that Nivre calls an oracle, which can not only predict the transition to be carried out, but also the dependency label for the Left Arc and Right Arc transitions (Nivre et al., 2007b). With an oracle to predict what transitions the system should make, the algorithm itself for deterministic dependency parsing is quite simple, applying a SHIFT transition if the stack is empty, and if the stack is not empty, enacting the transition specified by the oracle. There are several ways to approximate an oracle, for example by using a formal grammar, but the most successful approaches, and the approaches used by the systems examined in this thesis, use a data-driven approach, which approximates an oracle by training a classifier on treebank data. The task of this classifier is to predict the appropriate oracle transition for any given configuration, having been trained on feature vector representations of the configurations present in a gold-standard treebank. The models examined here differ primarily in how they go about representing these features, and the specific machine-learning methods employed for learning and classification. The subsequent sections provide a detailed overview of how these parser systems differ.

2.2.4 History-based feature models

Maltparser is an open-source data-driven dependency parser that was first launched in 2006, which uses inductive dependency parsing to derive the syntactic analysis of sentences, meaning that inductive machine learning is used to guide the parser at specific points in the parsing of a sentence (Nivre et al., 2006). This parsing process comprises three steps: a parsing algorithm for constructing the dependency structure, a feature model to predict what steps the parser should follow, and a machine learning algorithm to map the feature model to the parser actions. The primary point of differentiation between MaltParser and more recently developed neural network transition parsers is the history-based feature models that MaltParser uses to predict the actions of its parsing algorithms. These models use features of the input string together with features of the incomplete dependency structure. History-based feature models are used for a variety of tasks in NLP, and were first introduced by Black et al. (1992). In a history-based feature model, first a conditional probability for a pair \((x, y)\) comprising an input string \(x\) and an analysis \(y\) to a sequence of decisions \(D = (d_1, \ldots, d_{i-1})\) is calculated. The conditioning context for each decision is called the history, and typically corresponds to a partially completed structure. In the case of a deterministic parsing strategy, this partially completed structure refers to the possible decisions of the parser (the transitions). These histories are represented as sequences of attributes, or feature vectors, with the feature model determining the classification of these feature vectors. According to Nivre et al. (2007b), the most important features in dependency parsing are the attributes of input tokens, in this case the word form, or part-of-speech or dependency type.

The parser organizes these feature models around nine data structures. These data structures include: a stack (STACK) containing partially processed tokens, a list (INPUT) containing remaining input tokens, a stack (CONTEXT) of unattached tokens between STACK and INPUT, a function HEAD which defines the partially constructed dependency structure used to denote the syntactic head of a token, a function DEP used to label the dependencies in the developing dependency graph, a function LC used to define the leftmost child of a token, a function RC to define
the rightmost child of a token, and two functions LS and RS, used to denote the left sibling and right sibling of a token in the partially built dependency structure. Features (a word, part-of-speech or dependency type) can then be defined according to the relevant data structure (either STACK, INPUT or CONTEXT), using the functions HEAD, LC, RC, LS and RS.

In the feature model, each feature is represented on a single line, with columns denoting feature type, the type of data structure used in the configuration, with the subsequent columns filled by numerical values encoding information about the configuration. An example of the feature model as described in Nivre et al. (2006) can be found below:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Type</th>
<th>Data Structure</th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
<th>Column 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>POS</td>
<td>STACK</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>POS</td>
<td>INPUT</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>POS</td>
<td>INPUT</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DEP</td>
<td>STACK</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DEP</td>
<td>STACK</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

In this example, the first line is the part-of-speech of the token on the top of the stack (TOP). The second line is the part-of-speech immediately after the next input token in the input list (NEXT). The third line represents the part-of-speech of the token immediately preceding NEXT in the original input string (which may not still exist in the INPUT string or STACK anymore). The last two lines are the dependency type of the head of TOP (zero steps down the stack, zero steps forward/backward in the input string, one step up to the head) and the leftmost dependent of TOP (zero steps down the stack, zero steps forward/backward in the input string, zero steps up through heads, one step down to the leftmost dependent). The feature model must be designed specifically for each data set and language. Once the model is set, a learning algorithm can be used to map parser histories, following the feature model, to the behavior of the parser by means of a parsing algorithm. The most recent versions of MaltParser have two built-in learners: LIBLINEAR (Fan et al., 2008), and LIBSVM (Chang and Lin, 2011), although in this project we only make use of LIBLINEAR.

### 2.2.5 Parsing with neural networks

A recent trend in transition-based dependency parsing is exploring how neural networks can be used to improve the approximation of an oracle, and this trend has resulted in some of the most successful dependency parsers to date. One of pioneering efforts in this neural network-based parser enhancement was carried out by Chen and Manning (2014), in which the authors explore the potential benefits of using a neural network classifier for transition-based dependency parsing. In their study, the authors observe that prior dependency parsing systems use a large number of very sparse indicator features such as lexical features or word-class features, which not only pose a significant computational expense, but tend to generalize poorly. The authors note that this often because the feature weights are poorly estimated. The authors further note that while most previous feature-based discriminative parsers, MaltParser included, are quite effective, they could be improved by mitigating their reliance on manually designed feature templates, and the time-consuming process of feature extraction.
The authors of this study posit that these improvements can be attained by using a neural network that learns a dense and compact feature representation. To do this, they propose a way to encode all configuration information, features and interactions in as little as 200 dimensions. Then, a standard neural network, with a hidden layer and a softmax layer, is used for classification. They start building the feature representation by first using distributed representations to construct the features and interactions. Each word is represented as a d-dimensional dense vector, known as a word embedding. They extend their use of d-dimensional dense vectors to include POS tag and dependency label representations in what the authors call POS and dependency embeddings. They base this decision on the idea that, like lexical features, these smaller discrete sets also reflect a degree of semantic similarity (for example, NNS, a plural noun, is likely to be semantically similar to NN, a singular noun).

The next step in the construction of the feature representation is the selection of a set of useful words based on those words’ position in the configuration for each type of information (word, POS, or dependency label). This set is made up of the top three stack and buffer elements, the leftmost and rightmost children of the stack elements, as well as the leftmost of the leftmost/rightmost of the rightmost of the top two words on the stack. Lastly, the different embedding types (word, POS, or dependency label) for each useful word extracted from the configuration are concatenated together to form the final input layer for the neural network. The input layer is then mapped to the hidden layer through an activation function that the authors refer to as a "cube activation function", which they claim better captures the interaction of the three feature types used. Finally, the output layer is added, which maps the softmax probabilities from the hidden representations to the transitions. Figure 2.4 shows how the neural network classifier is designed.

![Neural network architecture for the Stanford Neural Network Dependency Parser](image)

During training, the authors use a fixed oracle to generate training examples, with a training objective of cross-entropy loss. All errors are back-propagated to the embeddings. Word embeddings are initialized by using pre-trained word vectors, while other non-word parameters are set through random initialization. During each step of the parsing process, tokens and their corresponding POS tags and dependency labels are extracted from the present configuration, and the hidden layer is computed through matrix multiplications. A pre-computation trick is used to further speed up parsing,
in which matrix multiplications are computed for the 10,000 most frequent words and stored in a table. This results in these calculations only needing to be performed once, and can be simple looked-up in the table for all subsequent instances. This pre-computation step is similarly carried out for POS tags and dependency labels. The authors believe that this step increased the speed of the parser by a factor of between 8 and 10.

Chen and Manning’s system improved labeled and unlabeled attachment scores (a standard evaluation metric for dependency parsing systems) by an impressive 2%, from 90.1% to 92.2%, although they noted that it could be possible to combine their neural network classifier with search based models to further improve accuracy. Weiss et al. built upon the model proposed by Chen and Manning, noting that while the Stanford Neural Network Dependency Parser outperformed similar greedy parsers with hand-engineered features, it was not competitive with state-of-the-art dependency parsers that are trained for structured search. Weiss et al. Expand upon Chen and Manning’s parser with three key contributions:

1. The use of "tri-training" to leverage unlabeled data.
2. Tuning the neural network.
3. Adding a structured perceptron with beam search to the training procedure.

Figure 2.5: Neural network architecture for the approach used by Weiss et al.

Figure 2.5 depicts the model design of the system proposed by Weiss et al. and serves to highlight the key changes made to Chen and Manning’s system. Because neural networks are known to require very large sets of training data, which can be difficult to come by for dependency parsing, they use a procedure called "tri-training" (Li et al., 2014), wherein they use two different parsers to automatically annotate a very large
amount of corpus data, and only select those sentences both parsers agreed on to be included in their final training set for the parser. Their work also differs in several other small ways to Chen and Manning’s, namely in that they use smaller embeddings for part-of-speech tags and labels, the use of Relu units in their hidden layers, and an addition of two hidden layers to their neural network. They motivate their addition of a structured perceptron with beam search by first stating a problem shared by all greedy algorithm-based parser models; that they are unable to look beyond one step ahead, and that they are incapable of recovering from incorrect decisions, leading to error propagation. Their proposed solutions is to look forward in the tree of possible transition sequences, keeping track of K top partial transition sequences up to a depth of m. Then, they score the transition with a perceptron. Weiss et al. observed a 1% increase in model performance, which they attribute to these modifications. Andor et al. (2016) further build upon the work done by both Chen and Manning (2014) and Weiss et al. by using global normalization instead of local normalization to address the label-bias problem often found with beam search. Label bias occurs in localized models that have a very weak ability to to revise earlier incorrect decisions. Andor et al. (2016) observe that the label bias problem implies that globally normalized models can be strictly more expressive than locally normalized models, and reported state-of-the-art accuracies for their system through their use of globally normalized models. In their paper, Andor et al. (2016) also announce the release of SyntaxNet which includes Google’s state-of-the-art dependency parser, Parsey McParseface, which is based upon the system presented in the paper. Additionally, they also announced the release of 40 so-called ”Parsey cousins” which were trained on Universal Dependencies treebanks (Nivre et al., 2016).

2.2.6 Evaluation metrics

In the previous section, we discussed the relative accuracies of various parser systems, without going in to detail as to how these percentages were derived. This raises the question as to how the accuracy of a parser is most often determined. The standard method of determining the accuracy of a parser is to reserve a test set in a treebank from the rest of the training data, use the parser to parse that test set, and then compare the output to the gold-standard annotations of the test set found in the treebank. Dependency parsers can be evaluated with several different metrics, the most popular of which, according to Kübler et al. (2009) are:

- **Exact match**: the percentage of parses that are completely correct (same metric most often used with constituent parsers).

- **Attachment score**: the percentage of words having the correct head. This is possible because of dependency parsing’s single-head property, which makes dependency parsing like a tagging task, wherein every word is to be tagged with a head and a dependency label.

- **Precision**: the percentage of dependencies with a specific label in the parser output that were correct

- **Recall**: the percentage of dependencies with a specific type in the test set that were correctly parsed.

- **F1 Score**: the harmonic mean of precision and recall.
It is possible to calculate all of these scores looking only at heads (unlabeled) or looking at both heads and labels (labeled), and the most common metrics are unlabeled attachment score (UAS) and labeled attachment score (LAS).

2.3 Annotation schemes for dependencies

There are various different formats for annotating dependencies, and in previous years it was common when developing a treebank for a specific language to pick an annotation scheme that was most appropriate for capturing that language’s grammatical structure. In recent years, however, there have been attempts to develop annotation schemes that do not favor one type of grammatical structure over another, so that they can be used to annotate multiple languages. This effort to create a versatile annotation scheme is particularly relevant to parser evaluation for several reasons. [McDonald et al. 2013] summarizes these reasons as follows: firstly, a representation that can be used for all languages is necessary for multilingual language technologies that require consistent cross-lingual analysis for downstream components. Secondly, consistent syntactic representations are ideal for evaluating unsupervised or cross-lingual parsers. The importance of a homogeneous representation for evaluation was demonstrated in a study by [McDonald et al. 2011], where delexicalized parsing models for several different source languages were evaluated on different target languages, the authors discovered that the highest scoring models were often not those trained on source languages that were typologically similar to the target. In one example, a model that had been trained on Danish was the lowest scoring parser for Swedish, due entirely to the differences in their annotation schemes.

To resolve these issues, [De Marneffe et al. 2014] propose a cross-linguistic typology called Universal Stanford Dependencies, which is based upon the Stanford Dependencies (SD) representation (De Marneffe and Manning 2008). The SD representation, which was developed specifically with English in mind, follows the lexicalist hypothesis in syntax, in which the grammatical relations are mapped between whole words. This is preserved in the Universal Stanford Dependencies representation. The authors also suggest a two-layered taxonomy: a more general set of basic relations that can be applied cross-linguistically, as well as a set of language-specific relations to allow for more comprehensive grammatical modeling. The Universal Dependencies project (Nivre et al. 2016) built upon the work of [De Marneffe et al. 2014], and merges ideas from several existing frameworks: the universal SD representation, the universal Google dependency scheme (McDonald et al. 2013), the universal Google part-of-speech tags (Petrov et al. 2012), and the Interset interlingua for morphosyntactic tag sets used in HamleDT treebanks (a project that converts treebanks to a standardized annotation scheme) (Zeman et al. 2014). The UD project intends to replace all other representations as the de-facto standard. The project’s philosophy is to provide a universal inventory of categories as well as guidelines for consistently annotating treebanks regardless of language. Like the universal SD representation, UD (version 1) comprises two layers of annotation, and follows a lexicalist approach to annotation. The scheme contains 17 part of speech tags and 40 grammatical relations between words. The grammatical relations distinguish between 3 types of structures: nominals, clauses, and modifier words. The scheme also distinguishes between core arguments (subjects and objects) and other arguments. The data is formatted according to the CoNLL-U guidelines, which are an evolution of the CoNLL-X format [Buchholz 2018].
and Marsi (2006), where each line represents one word/token, and comprises ten tab-separated fields which are used to specify, in this order: a unique id (integer for words, ranges for multiword tokens), word form, lemma, universal part-of-speech tag, optional language-specific part-of-speech tag, morphological features, head, dependency relation, additional dependencies in the enhanced representation and miscellaneous information. Blank lines delimit sentence boundaries, and lines beginning with the hash symbol are comments. Figure 2.6 is a sentence taken from the English UD treebank, and provides an example of CoNLL-U format.

![Figure 2.6: Example of CoNLL-U format, taken from the UD English Treebank](image)

### 2.4 Error analyses of dependency parsers

In order to improve parsers, compare parser systems, as well as to understand the limitations of dependency parsers with respect to how they perform differently on different languages and domains, researchers carry out error analyses to characterize the mistakes that these systems make. One of the earliest comprehensive error analyses for dependency parsing is described in the work by McDonald and Nivre (2007), in which they compare a graph-based parsing system MSTParser McDonald et al. (2006) to MaltParser, a local greedy transition-based parsing system. In their analysis, they found that while the parsers had similar performance in general, the two systems produced very different types of errors, with each system producing errors that could be explained by the theoretical properties of the model.

To carry out the error analysis, they used both parsers to parse data from 13 different languages, and carried out experiments that related parsing errors to a set of structural and linguistic properties of both the gold-standard test data as well as the parser output. First they examined length factors, by grouping the accuracies of both models by sentence length. In doing so, they found that, for shorter sentences, MaltParser, a greedy model, outperformed the graph-based MSTParser because shorter-sentences provided less opportunity for error propagation, and it has a richer feature representation than MSTParser. They also looked at accuracy relative to dependency length (the number of tokens between a dependent and its head), and similarly found that MSTParser was better equipped than MaltParser to handle longer dependencies, and MaltParser outperformed MSTParser for shorter dependencies. This can be explained using the same reasoning that applied to sentence lengths; that shorter dependency arcs are created before longer dependency arcs in MaltParser’s greedy parsing algorithm, and are as a result less prone to error propagation.

In addition to length-factors, they also investigated graph factors, namely, the accuracy of arcs relative to their distance from the root node, accuracies for what they call "sibling arcs", or dependencies that syntactically modify the same word, as well as accuracies relative to the degree of non-projectivity in a sentence. The
degree of non-projectivity is determined by the presence of non-projective arcs in a sentence, as well as the length of these non-projective arcs. When looking at these graph factors, they found that MaltParser was more precise than MSTParser for arcs farther away from the root. They theorize that this is because arcs further from the root are constructed early in the parsing algorithm for MaltParser. They also observed that MaltParser has a tendency to over predict root modifiers, because all words that the parser fails to attach are attached automatically to the root. As a result, they note that low precision for root modifiers is indicative of a transition-based parser producing fragmented parses. In the case of sibling arcs, McDonald and Nivre (2007) found that MaltParser was slightly less precise for arcs that more siblings, possibly due to the fact that words with a greater number of modifiers tend to be closer to the root. They also found that MaltParser was less precise in predicting sentences with a high-degree of projectivity, presumably because such sentences tend to be correlated with longer dependency arcs, distance to root and number of siblings.

In addition to length and graph factors, the authors also discussed linguistic factors’ role, such as parts of speech and dependency types in the comparative accuracy of the models. In doing so, they found that MaltParser was more accurate for nouns and pronouns, while MSTParser performed better for all other categories, especially conjunctions (and in particular when coordination spans a considerable distance). This is in keeping with their findings for length and graph factors, in that verbs and conjunctions most often make up parts of dependency relations that are closer to the root and span longer distances, while nouns and pronouns are attached to verbs and, as a result, often occur lower in the graph. Because MaltParser performs better for nouns and pronouns, it has an advantage for identifying nominal categories, and for identifying subjects in particular.

In a follow up study, Nivre et al. (2010) evaluated the same two parsers, MSTParser and MaltParser on their ability to handle unbounded dependencies in English. They define an unbounded dependency as one involving a word or phrase interpreted at a distance from its surface position, where an unlimited number of clause boundaries may in principle intervene. To evaluate the extent to which each parser recovered these unbounded dependencies, the authors tested the parsers on the unbounded dependency corpus developed by Rimell et al. (2009), which contains 700 sentences annotated with unbounded dependencies. The corpus contains seven different types of grammatical constructions: object extraction from a relative clause (ObRC), object extraction from a reduced relative clause (ObRed), subject extraction from a relative clause (SbRC), free relatives (Free), object questions (ObQ), right node raising (RNR), and subject extraction from an embedded clause (SbEm). Nivre et al. (2010) found that labeled attachment scores for both systems were quite low compared to the each system’s highest reported LAS for English. They also observed similar trends to McDonald and Nivre (2007), noting that MaltParser tended to outperform MSTParser on unbounded dependencies where argument relations tend to be relatively local (e.g. relative clauses), while MSTParser was better equipped to parse more distant relations. In general, MaltParser performed better on ObRC, ObRed, SbRC and Free dependencies, all of which are more local, while MSTParser did better on dependencies such as ObQ, RNR and SbEm, which are more global. While the two systems differ significantly in the types of errors they make, the overall accuracy of both systems is quite similar.

More recently, de Lhoneux et al. (2017), carried out a comprehensive error analysis on MaltParser and UDPipe, a neural network parser, in an attempt to characterize
the differences in errors made by classical transition-based parsers compared to newer neural network-powered transition parsers. To do this, the authors used both parsers to parse data from 8 different languages, using treebanks from the Universal Dependencies project. First, the authors compared the systems based on overall accuracy, finding that UDPipe outperformed MaltParser on nearly all languages, but not for those that only had very small treebanks. The authors also investigated the effect of training size on neural network parsing, with the hypothesis that the neural network parser has a much steeper learning curve than MaltParser. They found however that this hypothesis was only partially correct: while they observed a steep learning curve initially, once treebank sizes surpassed several thousand tokens, the learning curve flattened and the model continued to improve at similar rates to MaltParser. Interestingly, they also found that MaltParser failed to outperform UDPipe on all treebanks with a training size of only one thousand tokens.

For their error analysis, the authors largely followed the criteria laid out by McDonald and Nivre (2007), comparing the systems’ respective errors with regard to graph and linguistic factors. While they found that the differences between the MaltParser and UDPipe were not as clear cut as they were between MaltParser and MSTParser, the same general trends seem to hold. That is to say, like for McDonald and Nivre (2007), MaltParser fared worse than its competitor for longer dependencies. This same trend did not hold for sentence length, however, where, while MaltParser was outperformed by UDPipe for sentences between 1-10 tokens, and 20-30 tokens, but performed similarly to UDPipe for sentences 30-40 tokens, before eventually being overtaken again for sentences over 50 tokens. When looking at linguistic factors, the authors looked at the 15 most frequent dependency relations, and again found results that echoed the findings by McDonald and Nivre (2007). They found that MaltParser has a slight edge on consistently shorter dependencies such as nominal modifiers, with UDPipe clearly better equipped to handle distant dependencies. In general, MaltParser only outperformed UDPipe for a limited number of specific phenomena all involving short dependencies.

Other studies have taken a broader approach to parser evaluation, comparing a variety of dependency parsers representing many different approaches to parsing, rather than only two types of parsers. Choi et al. (2015) developed a web-based tool for helping non-experts compare “off-the-shelf” parsers, and used it to compare ten state-of-the-art dependency parsers. This allowed them to compare greedy vs. non-greedy parsers, as well as a variety of transition-based parsing systems and graph-based systems on seven different genres of text. They then evaluated the systems according to speed and accuracy. They then analyzed the systems’ accuracy by sentence length, dependency distance, non-projectivity, POS tags and dependency labels, and genre. They also carried out a manual error analysis focusing on sentences that proved problematic for multiple parsers.

When examining overall system accuracy, they found that Mate parser (Bohnet, 2010), a graph-based parser performed best, although the differences in accuracy between Mate parser and the other top five performing parsers were quite small. When they tested for speed, they found that transition-based parsers outpaced their graph-based counterparts. When they compared parsers across genres, they found that accuracy was highest for pivot text and the Bible, while accuracy was across all systems was lowest for more the informal genres: telephone conversation and web data. When they performed an in-depth error analysis, they found that the Stanford Neural Network Parser had the highest number of what the authors coin "uniquely
bad parses, which largely seemed to be a product of the greedy algorithm choosing the first reasonable choice of root, which led to frequent root errors, leading to error-propagation. To further analyze their results, they also looked at the texts themselves to find potential sources of error. When analyzing the data, they found that verbs (either lack thereof, or there existing several verbs in a sentence) were frequent sources of parser error, as well as fragmented sentences. They also found that typographical errors, sentence initial discourse cues, sentence segmentation problems (i.e. one sentence containing multiple sentences), disfluency in the sentence to cause significant parser confusion.

Hashemi and Hwa (2016) focus their study on parser robustness for ungrammatical sentences, comparing eight state-of-the-art dependency parsers on two domains of ungrammatical sentences: learner English and machine translation outputs. They train their parsers on a treebank of standard English, the Penn Treebank [Marcus et al. (1993) and Tweebank [Kong et al. (2014)] and then evaluated both parsers on two datasets, one containing sentences written by non-native speakers, and one containing machine translation outputs. The study revealed that when trained on standard English data, graph parsers outperformed transition-based parsers. However, when trained on the smaller Tweebank corpus, MaltParser performed best on learner English data, and despite the small size of the training set, SyntaxNet outperformed all other systems on the machine translated data.

2.5 Domain adaptation for parsing

In natural language processing, the notion of domain can be loosely defined as a collection of texts or genres often displaying similar linguistic patterns and characteristics, such as newswire text, web text, or biomedical journals. Often in statistical parsing, systems that are trained and tested on the same domain test quite highly, but their performance suffers when the same system is used to parse out of domain data. In particular, most research in dependency parsing has focused on the Penn Treebank and English Web Treebank (Silveira et al., 2014), which are more biased toward more formal written English than they are toward informal and conversational language. As a result, in order for such systems to be of practical use in more diverse settings, they must be adapted to suit other domains.

Baucom et al. (2013) divide domain adaptation for dependency parsing in to two different scenarios: one where a limited amount of annotated data is available, and one where no data from the target domain is available. They note that not having access to any annotated target data at all tends to make domain adaptation especially challenging. This was exemplified by the CoNLL 2007 shared task on dependency parsing [Nivre et al. (2007a)], in which no team’s results for the out of domain setting improved much over the baseline. For the task, participants were provided with an annotated treebank for the source domain as well as unannotated data from three target domains: biomedical abstracts, chemical abstracts and parent-child dialogues. They were then asked to the best possible parsing results across all three domains. Results were to be reported in labeled attachment scores. The most successful system used an ensemble-based approach (Sagae and Tsujii) in which they used a combination of two different models of an LR parser and then selected only output for which both systems agreed to add to their final training set. Other submissions to the shared task used rule-based approaches for tree revisions or filtering the the training set based on
similarity metrics to the target domain. Other attempts have involved self-training, such as McClosky et al. (2006), in which they combined self-training with a re-ranker to reach an error reduction of 28%. In self-training, a parser that has been trained on the source domain is used to parse unlabeled data in the target domain. Then, that automatically annotated target domain data is used as training data. This process can be iterated multiple times until parsing accuracy fails to improve at each iteration.

Reichart and Rappoport (2007) show that self-training can improve a model without the need for reranking provided access to a small amount of annotated in-domain seed data. They train a parser on this small data set, and use it to create a larger training set of automatically annotated data. They measure their success by annotation cost, and at evaluation found that their system resulted in a 50% reduction in annotation cost. Chen et al. (2008) opted for a different approach, and developed a system that requires no annotated seed data. For their approach, they use a variant of self-training in which instead of selecting full dependency trees for their second training step, they add only short dependencies, which tend to have a much higher accuracy rate, to their set of automatically parsed training data. This resulted in a 1% improvement in LAS for Chinese.

Another method tried for domain adaptation is uptraining, coined by Petrov et al. (2010), in which a deterministic parser is trained on a slower, but more accurate constituent parser. The authors used a constituent parser to parse questions, converted the output to dependencies, and then used the output to train a dependency parser. They found that uptraining with 100,000 unlabeled questions produced comparable results to training on 2,000 manually annotated questions, and was able to bring parsing accuracy on questions to 84%, which is on par with in-domain parser performance.

Baucom et al. (2013) attempted co-training, where they used two different parsers each with a different parsing algorithm (with different biases). The parsers are used to parse unannotated data, and provide a confidence score with each parse. The output of each parser is ranked by confidence score, and the n-best parses from each parser are added to the original training data. Then the process is repeated again on fresh unlabeled data. This process is repeated until no further improvement on the development set is observed. Baucom et al. (2013) observed that co-training raised parser accuracies approximately 1% above a baseline. In the same study, the authors also attempted parser combination, similar to the approach used by Sagae and Tsujii. They tested ensembles of 3 parsers and ensembles of 2 parsers, and found that this method reach similar improvements over a baseline to co-training, although it required very large amounts of training data, while co-training was successful with only a very small amount of data.

2.6 Dialogue systems and conversational agents

Dialogue (between humans or otherwise) is characterized by turn-taking, in which speaker A says something, and speaker B responds, and so on, and is typically made up of speech acts, such as asking or answering a question, agreeing with a statement, etc. A dialogue system is a program that communicates with users in natural language, often to solve a task such as booking travel arrangements or accommodations, answering questions about the news or weather, acting as a virtual
assistant (for example by saving calendar appointments or calling a contact), or even more sophisticated tasks. Figure 2.7, taken from Jurafsky and Martin (2009) provides a simple example for the layout of a dialogue system. This example shows a spoken dialogue system that has six components. The speech recognition and natural language understanding components derive meaning from user input, and the natural language generation and text-to-speech (TTS) synthesis components produce an appropriate response to the input. It is the job of the natural language understanding component to produce a semantic representation of the input that will aid the system in solving the dialogue task. A common way of doing this is based on frame-and-slot semantics, in which a "frame", which corresponds to a topic has "slots" for details about that topic (Jurafsky and Martin, 2009).

For example, the sentence *Show me morning flights from Boston to San Francisco on Tuesday* could be represented with the frame *flights* containing slots with information about the city of origin, the date of departure, the time of departure, the destination city, etc. To produce these representations, such systems often rely on semantic grammars, a type of context free grammar. The dialogue manager controls the architecture and structure of the dialogue, and the task manager is capable of providing knowledge about the task. A common way of constructing the dialogue manager is finite-state automata (FSA), in which the states of the FSA correspond to
questions that the dialogue manager poses to the user, and the arcs between the states are the various actions to take depending on the user response. Figure 2.8 provides an example of a finite state dialogue manager (Jurafsky and Martin, 2009).
3 The Dataset

In order to fulfill our objective of assessing parser accuracy on dialogue data, we first collected a dataset of 882 sentences from various human-computer dialogue interactions on which to test the parsers. The data were collected from 12 different Artificial Solutions projects, and represent data from various different dialogue “genres”, so to speak. That is to say, while some sentences are the result of interactions with customer support chat bots, others have been taken from conversations with virtual assistants, which may involve an automatic speech recognition (ASR) pre-processing component, and some sentences stem from chatterbot conversations, in which a program simulates a conversation that can cover any number of topics, and are often used to showcase dialogue systems. This makes our dialogue data itself quite diverse; the customer support conversations feature a large number of questions and are often limited to a set list of topics (i.e. if a customer is contacting an airline, it is likely the customer’s question will be about a flight), the virtual assistant interactions tend to contain a mix of commands (e.g. Set my alarm for 7 a.m.) and questions (e.g. Is it going to rain today?) and may contain ASR transcription errors, whereas the chatterbot conversations tend not to correspond to any one type of sentence structure or topic in particular. Furthermore, as the chatterbot serves no one specific function and is something of a novelty application, the chatterbot data contains informal language such as expletives, sentence fragments, foreign words, etc.

3.1 Annotating the dataset

In order to evaluate parser accuracy on our data, we needed to produce a gold standard annotation of our data for comparison. This task was carried out manually by a native English speaker, who was first given a week to familiarize herself with the Universal Dependencies guidelines. The annotator is a master’s student in Language Technology with a background in linguistics, and has previous experience working on the Universal Dependencies project. The annotations were produced in the CoNLL-U format (as per UD guidelines) and were marked using a web application called Conllu Editor, which is an online tool for editing files in the CoNLL-U format. The annotator marked the dependency relations as well as the UD part-of-speech tags. The data set had already been manually part-of-speech tagged with Penn Treebank POS tags. In order to ensure consistency of annotation with UD guidelines and with the English UD treebank, given the lack of inter-annotator review, the annotator also used the English UD treebankSilveira et al. (2014) as a guide while annotating, using regular expressions to search for commonly occurring phrases and part-of-speech tags and their corresponding most frequent syntactic annotations. For example, if the
annotator was deciding how to designate the relation between the words "Washington" and "D.C." in the city name "Washington, D.C.", the annotator would search the treebank for the full string "Washington, D.C.", and find that in all cases, "D.C." is the dependent of "Washington" with the label "appos". If the search failed to return an exact string match, it could be extended to syntactically similar phrases (e.g., "Dallas, Texas") to find similar dependency relations to use as a guide.

Annotating the dataset proved challenging for several reasons, most of which stemmed from the informal tone of many sentences, which led to a certain degree of ambiguity. In many cases, especially in sentences relating to customer support, this was caused by users of the dialogue system using a form of shorthand to quickly communicate a problem or ask a question. As a result, there are some sentences for which we parse the sentence literally based on the input provided by the user. For example, the sentence video not working green screen with no audio we chose to interpret as The video is not working: I get a green screen with no audio and was annotated in Conllu Editor as is shown in Figure 3.1. This relation between screen and the root is annotated as parataxis because it is the closest possible dependency relation to the surface representation of the sentence. While it is possible to fill in the blank between working and green with a subordinating conjunction, we viewed this interpretation to be too abstracted from the original sentence. For other sentences,

| 1 | video | video | NOUN | MN | 3 | noun |
| 2 | not | not | RB | RB | 3 | neq |
| 3 | working | work | VBD | VBD | 0 | NOOS |
| 4 | screen | screen | NOUN | N | 5 | anmod |
| 5 | with | with | ADP | IN | 7 | case |
| 6 | audio | audio | NOUN | N | 7 | anmod |

Figure 3.1: Example annotation of an ambiguous sentence with parataxis in Conllu Editor

however, the missing relation was more clearly implied, for instance in cases of omitted prepositions. Other annotations of non-standard English, most notably Berzak et al. (2016), who developed a treebank for learner English, opt for a literal treatment of missing prepositions, and annotate nominal dependents of a predicate that are missing a preposition as arguments rather than nominal modifiers. For our purposes, we decided against this annotation choice, because in our data set prepositions were systematically dropped, leading us not to interpret this a grammatical error, but rather a feature of this domain. Consider Figure 3.2 which shows the annotation for the sentence change of address joint account, which we have interpreted as change of address on a joint account. We suspect that the distinction between these two types of sentences will prove especially challenging during parsing, given that on the surface the sentences in Figures 3.1 and 3.2 are very similar, and it is only using semantic cues that we are able to infer the differences in their syntactic structures.

In the case of typographical errors, we annotated them according to the perceived intended meaning of the user, for example the sentence I recognize to general properties was annotated with to read as two as shown in Figure 3.3.

For some sentences, it was difficult to decide on one preeminent syntactic interpretation of the sentence. For example, in the sentence featured in Figure 3.4 it is difficult to say which is the main clause in the sentence, and how the clauses relate to one another. Without any clear semantic connection between the two sentences,
Figure 3.2: Example annotation of sentence with an omitted preposition and determiner

we label the relation as parataxis, but there is also room for debate on exactly where the two clauses diverge: is animals an argument of think or is think the root of the subsequent clause? Does the phrase in the last 4000 years modify stick or it? Because the sentence itself has very little meaning it is difficult to make these decisions. As a result, we would argue that some parser deviations from the gold standard should not necessarily be considered errors, because for some sentences there is no one obvious correct parse.

Figure 3.3: Example annotation of a typographical error

After annotating the data set, we used the python validation script provided by the Universal Dependencies project to confirm that our annotations complied with UD guidelines. We then manually checked the trees for errors using the MaltEval tree viewer (Nilsson and Nivre [2008]). Because one of the parsers we tested only outputs Stanford Dependencies, we also manually converted our Universal Dependencies annotations to the Stanford Dependencies format. Following this conversion, we automatically checked for any lingering unconverted Universal Dependencies that we missed and corrected them.

3.2 Analyzing the dataset

Once our data were annotated, we analyzed the data set to characterize the ways in which it differs from standard English treebanks for which we took the (representative)
English UD treebank. To gain a more comprehensive understanding of the features that define this data set, we look at factors such as sentence length and dependency arc length, as well as the relative distributions of POS tags and dependency labels. We also examine other linguistic characteristics of the data set, such as the relative frequency of questions, sentence fragments, grammatical errors as well as typographical errors and misspellings. We also performed a statistical analysis of the English UD treebank to allow us to make some generalizations as to how our data set differs from other standard English treebanks.

Table 3.1 shows the mean and median sentence and arc lengths in both our data set (DTS) as well as in the English UD (EN-UD) treebank. As the table shows, on average, sentences in the EN-UD treebank are nearly twice as long as those found in our test set. Additionally, the average arc length is considerably greater in the EN-UD treebank, which is in keeping with expectation for longer sentences. It is worth noting, however, that the median arc length for both data sets is the same, although the same cannot be said for the median sentence lengths. Because, as we discussed in section 2.1, it has been shown that shorter sentences and shorter dependencies tend to be easier to parse, especially for transition-based parsers, we expect that the shorter sentences and arcs in our data set should make it, at least in this regard, easier to parse than other standard English treebanks.

<table>
<thead>
<tr>
<th></th>
<th>M sentence length</th>
<th>Mdn sentence length</th>
<th>M arc length</th>
<th>Mdn arc length</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN-UD</td>
<td>16.32</td>
<td>14.00</td>
<td>3.52</td>
<td>2.00</td>
</tr>
<tr>
<td>DTS</td>
<td>9.23</td>
<td>8.00</td>
<td>2.19</td>
<td>2.00</td>
</tr>
</tbody>
</table>

**Table 3.1: Mean and median sentence and arc lengths for our Dialogue Test Set (DTS) as compared to the English UD treebank (EN-UD)**

When we looked at the relative distributions of different POS tags in our data set and EN-UD treebank (specifically the training set of the treebank), we found that our data contained nearly no punctuation (making up approximately 0.6 percent of all POS tags), so we have provided statistics for the EN-UD treebank both factoring in and out punctuation. Figure 3.5 shows the percentages of different Universal Dependencies (UD) POS tags, a full list and description of which can be found at[^1]. As the data show, even when we remove punctuation from our analysis, there are some salient differences between the data sets. Firstly, while both the DTS and the EN-UD treebank have a similar number of nouns, the dialogue data features considerably more pronouns. However, contrary to our expectations, our data set actually contains fewer proper nouns than the EN-UD treebank. We also observe some patterns that we believe relate to the differences in sentence lengths between the data sets, namely the increased relative frequency of verbs, and the decreased frequency of subordinating conjunctions in our data. We also observe some features that we believe correspond to an increased frequency of questions in our data as compared to the EN-UD treebank, namely the higher frequency of auxiliary verbs which, barring some exceptions, are necessary for forming interrogatives in English. Some POS tag frequencies echoed observations we made while annotating the data, namely that determiners and prepositions are relatively infrequent as compared to the EN-UD treebank, which is consistent of the systematic preposition and determiner omission we observed earlier.

[^1]: [http://universaldependencies.org/](http://universaldependencies.org/)
We also measured the relative frequencies of dependency types in both data sets. In many ways, these distributions reinforce what we observed for the POS relative frequencies, as well as provide some insight into how our data set differs syntactically from the English UD treebank. Figure 3.6 shows the 15 most frequent dependency relations in our dialogue test set (DTS) and their corresponding relative frequencies in the EN-UD treebank. One of the most striking observations is how much more frequent the "root" label is in the DTS as compared to the EN-UD treebank. This presumably relates to sentence length, and we expect that a parser's ability to correctly identify the "root" label will have a strong correlation to overall parser accuracy [McDonald and Nivre (2007)]. We also observe that our data set contains a higher frequency of predicate arguments, with nominal subjects making up 12 percent of all dependency types in the DTS as compared to 9 percent in the EN-UD treebank (when we include punctuation this percentage drops below 8 percent), and direct objects representing approximately 8 percent of the DTS compared to roughly 5 percent in the EN-UD training set.

Compounds also make up a greater percentage of dependency types in our data set than in the EN-UD treebank, which is intuitive at least for the customer service domain (compounds relating company products and services are quite common, e.g. CompanyX customer service hotline wait times). We expect that such compounds will be a frequent source of parser error, as they often comprise various parts-of-speech, and even if the parser is able to correctly identify the entire unit as a compound, it must also correctly identify all dependency relations within the compound itself [Nakov, 2013]. We observe again that prepositions and determiners are relatively infrequent in our data set, and that the percentage of auxiliaries is quite high, occurring twice as often in the DTS as they do in the EN-UD treebank. We believe that this in conjunction with the fact that negation occurs more than twice as frequently in our data set is reflective of the greater relative amount of questions in our data set compared to the EN-UD treebank, given that during annotation time we observed that many of the questions concerned products that were malfunctioning (e.g. Why isn't my printer working).
3.3 Categorizing the data set

In the final stage of our analysis of the data, we categorized each sentence in our data set according to characteristics that we deemed germane to this domain. Sentences were categorized manually during annotation, and can fall into 3 different categories: questions (which also contained the subcategory embedded questions), fragments, and non-standard usage. We define a sentence as housing an embedded question when it follows or is preceded by an independent clause. These questions are typically dependent on the root through parataxis, as seen in figure 3.7.

Embedded questions differ from other questions in that they can be separated from the rest of the sentence and leave two standalone sentences. This is not the case for simple questions, as shown in Figure 3.8.

We categorize sentences missing either a subject or a predicate as sentence fragments. This definition does not extend to pronoun dropping in imperative constructions (e.g. *Cancel my subscription*), but does cover pro-drop in informal constructions such as *Would like to cancel my subscription*. Most fragments in our data set are nominal fragments, e.g. *new telephone number telephone banking*. We also account for sentences containing what we broadly refer to as non-standard usage, which covers typographical errors, misspellings, grammatical errors and non-standard word order.

Figure 3.9 shows the results of our analysis. We discovered that questions (em-
When I turn on my laptop why is the screen black.

Figure 3.8: Sentence in which a question constitutes main clause

bedded or otherwise) made up nearly half of our entire data set. As it has been demonstrated that questions tend to be more difficult to parse (Petrov et al., 2010), we expect this to have significant implications on parsing results for our data. Non-standard usage and fragments can be found in 10 percent and 12 percent of sentences, respectively, which we likewise expect to negatively affect parsing accuracy, as we presume these characteristics are less common in our training data.

Figure 3.9: Distribution of various sentence types in DTS

After discovering how question-oriented our data set is, we set out to estimate the relative frequency of questions in the English UD training set to provide a point of comparison. We accomplished this by searching the treebank for markers that are typically associated with questions, such interrogative adverbs and auxiliary verbs occurring in sentence-initial positions. To test the success of our question identification method, we applied it to our data set, in which we had already manually identified questions. We found that our method has quite a high precision of 98.2%, although recall was considerably lower at 85.6%. We assume that some of the recall loss can be attributed to this method’s inability to identify embedded questions, as it only looks for questions in the root clause. When we applied our search to the EN-UD training data,
we identified 471 question sentences out of a total 12,022 sentences, which puts the relative frequency of questions in the EN-UD treebank at around 3.9%. If we estimate that it is possible that we missed up to 20% of all questions, it is not unreasonable to assume that the total percentage of all questions in the EN-UD training data does not exceed far beyond 5%.

3.4 Discussion

Our observations about the data set allow us to make some predictions about what aspects of this domain will prove challenging for parsing. Firstly, we hypothesize that ambiguity caused by missing subjects, predicates, and prepositions will be a frequent source of parser error on the data set. Secondly, we expect that the high frequency of compounds in this domain will also make parsing difficult. We also expect that the high frequency of questions will further impede parsing, given that we know questions are more difficult to parse in general, and they are relatively rare in the training data. We further hypothesize that the significant differences in sentence and arc length in the test data vs. the EN-UD treebank data will make a parsing model trained on the EN-UD treebank ill-suited to the dialogue test set.
4 Parsing the data

4.1 Overall model accuracy

We tested MaltParser, which we trained on the EN-UD treebank using default settings, as well as two different SyntaxNet parsers, Parsey McParseface and the English "Parsey Cousin", which was trained on the UD English treebank. We wanted to find out about how Parsey McParseface compared to its UD trained cousin for several reasons. Firstly, it was trained on a larger corpus containing more data similar to the dialogue system domain than the English UD treebank. Secondly, because this model reported an LAS of 94.8% on questions, and 89.9% on web text (Andor et al., 2016), both of which bear many similarities to our domain, we were interested to see how it fared on our test set. We ran all three parsers on the dialogue test set using both human and automatically annotated POS tags. We used the Parsey Universal POS tagger to produce the automatic POS tags for MaltParser and the Parsey Universal dependency parser, and Parsey McParseface to generate the automatic POS tags for the Parsey McParseface parse test. We also evaluated MaltParser and Parsey Universal on the English UD test set. Because the corpus used to train Parsey McParseface is not publicly available, we rely on the accuracy scores reported in [Andor et al. (2016)] for comparison, although LAS and UAS are only reported for predicted POS tags.

<table>
<thead>
<tr>
<th></th>
<th>LAS\textsubscript{gold}</th>
<th>UAS\textsubscript{gold}</th>
<th>LAS\textsubscript{auto}</th>
<th>UAS\textsubscript{auto}</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaltParser\textsubscript{DTS}</td>
<td>80.0</td>
<td>85.2</td>
<td>70.7</td>
<td>78.5</td>
</tr>
<tr>
<td>Parsey Universal\textsubscript{DTS}</td>
<td>78.3</td>
<td>84.5</td>
<td>75.3</td>
<td>81.9</td>
</tr>
<tr>
<td>Parsey McParseface\textsubscript{DTS}</td>
<td>78.8</td>
<td>84.3</td>
<td>76.9</td>
<td>82.2</td>
</tr>
<tr>
<td>MaltParser\textsubscript{EN-UD}</td>
<td>81.1</td>
<td>84.4</td>
<td>72.3</td>
<td>78.4</td>
</tr>
<tr>
<td>Parsey Universal\textsubscript{EN-UD}</td>
<td>83.3</td>
<td>86.2</td>
<td>78.4</td>
<td>82.9</td>
</tr>
</tbody>
</table>

Table 4.1: LAS and UAS for DTS and EN-UD treebank with gold and automatic POS tags for MaltParser, Parsey Universal and Parsey McParseface

As Table 4.1 shows, MaltParser was the highest performing model, although only when using gold part-of-speech tags. When using predicted POS tags, Parsey McParseface produced the highest labeled and unlabeled attachment scores, although its edge over Parsey Universal is rather small, especially for UAS. We also note that Parsey McParseface’s LAS on the Union Treebank used by [Andor et al. (2016)] are not replicated on our test data. While both SyntaxNet models clearly outperform MaltParser with automatic POS tags, MaltParser’s considerable edge over SyntaxNet when using gold part-of-speech tags is interesting, particularly because this edge does not extend to the EN-UD test set, on which Parsey Universal outscored MaltParser across all metrics. We also observe that the SyntaxNet models in general have a narrower gap...
between automatic and gold POS tags than MaltParser. For the SyntaxNet models, this discrepancy never exceeds 5 percentage points, while for MaltParser the difference is nearly 10 percentage points for both the DTS and the EN-UD test set. The subsequent sections provide further insight as to why this is the case.

4.2 Detailed accuracy analysis

Sentence length

Figure 4.1 shows the LAS for all three parsing models when using gold and predicted POS tags. Here, we see MaltParser’s performance gap between gold and automatic POS tags even more clearly. For sentences between lengths of 3 and 25 MaltParser shows a clear and considerable disparity in this regard, with performance particularly dropping for longer sentences on the test set with automatic POS tags. While Parsey Universal’s output also shows a gold/automatic POS performance gap, albeit less severely, Parsey McParseface’s output is surprisingly consistent, showing only a small gap between gold and predicted part-of-speech tags for shorter sentences, with the model producing more accurate parses for the automatically part-of-speech tagged data set for some sentence lengths. Because this performance gap exists for the SyntaxNet UD model but not Parsey McParseface, it is possible that the POS tag set plays a role in this: Parsey McParseface outputs Stanford Dependency part-of-speech tags and dependencies, so it is possible that various factors relating to the annotation scheme are responsible for Parsey McParseface’s apparent advantage.

![Figure 4.1: Accuracy by sentence length for MaltParser and Parsey Universal (with gold and automatic POS tags)](image-url)
Arc length

We observe similar trends when we measure accuracy by the length of dependency relations, which is defined as the number of tokens between a head and its dependent. Firstly, we see that accuracy is higher for shorter dependencies, and tends to worsen for longer dependencies. We also see that MaltParser struggles with longer dependencies more than the SyntaxNet models, and while having an edge on the gold POS test set, this performance advantage drops off for dependency relations longer than 6 tokens. This is similar to the findings of de Lhoneux et al. (2017), in which they showed that MaltParser was less accurate for longer dependencies than a neural network-based parser. MaltParser’s considerable performance gap on the gold vs. automatically part-of-speech tagged data sets is again evident. For arcs of lengths greater than 11, the data are too sparse to make any generalizations about the performance of the three models.

![Figure 4.2: Accuracy by arc length for MaltParser and Parsey Universal (with gold and automatic POS tags)](image)

Accuracy by POS label

When we look at accuracy by POS type, it gives some insight into not only the gold/automatic POS performance gap, but also the relative performance of the different models in general. Here, we only compare the two UD models, because the shared tag set allows us to draw direct comparisons between the accuracy of both systems. As seen in Figure 4.3, MaltParser outperforms the SyntaxNet model for nearly all POS tags on the gold tagged data set, with accuracy dropping significantly on the automatically tagged data set. Again, we see that SyntaxNet’s accuracy is relatively stable for both data sets. We notice that for MaltParser, the gold/predicted POS accuracy disparity is particularly pronounced for adjectives, nouns (including proper nouns), subordinating conjunctions and verbs. For the automatic POS tag data set, SyntaxNet outperforms MaltParser for nearly all parts-of-speech, with the exceptions
of particles, interjections and numbers. When parsing the data set with gold part-
of-speech tags, SyntaxNet succeeded over MaltParser for conjunctions, punctuation and determiners. MaltParser’s higher accuracy on adjective and nouns we suspect comes from MaltParser’s ability to better distinguish between compound attachment to nouns and adjective attachment to nouns, at least when parsing with highly accurate POS tags. We also see that when parsing with gold POS tags, MaltParser has a much higher accuracy for adverbs, which could indicate this system’s higher accuracy in parsing questions, which would account for its higher overall performance, given that questions are so prevalent in our data.

![Figure 4.3: Accuracy by POS tag for MaltParser and Parsey Universal with gold and automatic POS tags](image)

**Figure 4.3** Accuracy by POS tag for MaltParser and Parsey Universal with gold and automatic POS tags

**Accuracy by dependency label**

We also tested the accuracy of both systems by dependency type, as seen in Figure 4.4 where we have listed the LAS for the 15 most frequent dependency relations in our data set. Here, the results were somewhat more consistent for both systems than for POS tags, but some trends carry over. Again, we see that MaltParser has higher accuracy scores on the gold POS tag data set, and lower scores on the automatic POS tag data set, with the gap being particularly pronounced for nominal possessive modifiers, open clausal complements, adjectival modifiers and compounds. We suspect that in the case of the latter two, MaltParser’s edge on the gold POS tag data set comes from its ability to more accurately parse compound nouns, which is made much more difficult with less accurate part-of-speech tags. For the gold POS tag data set, SyntaxNet is only more successful than MaltParser at handling markers, although it scores on par with MaltParser for determiners. We also notice that generally, all models struggle to parse nominal modifiers, which we see as symptomatic of the frequency of omitted subjects and prepositions in the data set.
Figure 4.4: Accuracy by dependency relation for MaltParser, Parsey Universal with gold and automatic POS tags

Accuracy for questions

Because questions make up such a significant portion of our data set, we wanted to find out how they affected overall parser performance, especially because it has been shown that parser accuracy tends to suffer when parsing questions. To test how our parsers handled questions, we isolated all questions in our data set and tested MaltParser and the UD SyntaxNet Parser, Parsey Universal, on this selection of the data set. We further identified all sentences containing embedded questions from this subsection of our data set, to see how the parsers fared on this specific type of questions compared to questions in general. Contrary to our expectations, MaltParser’s LAS only dropped slightly for questions, and UAS actually improved by half a percentage point, except in the case of MaltParser’s output on the version of the data set with predicted part-of-speech tags, for which scores dropped considerably. Parsey Universal’s performance dropped slightly for both the gold and automatic POS tagged versions of the data set, with its performance on the automatically tagged version particularly affected. For embedded questions, however, accuracy dropped for both systems, suggesting that these sentences pose a considerable challenge to parsing.

<table>
<thead>
<tr>
<th></th>
<th>All questions</th>
<th>Embedded Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LAS</td>
<td>UAS</td>
</tr>
<tr>
<td>MaltParser&lt;sub&gt;gold&lt;/sub&gt;</td>
<td>79.4</td>
<td>85.6</td>
</tr>
<tr>
<td>Parsey Universal&lt;sub&gt;gold&lt;/sub&gt;</td>
<td>77.0</td>
<td>83.0</td>
</tr>
<tr>
<td>MaltParser&lt;sub&gt;auto&lt;/sub&gt;</td>
<td>67.7</td>
<td>76.0</td>
</tr>
<tr>
<td>Parsey Universal&lt;sub&gt;auto&lt;/sub&gt;</td>
<td>73.8</td>
<td>81.0</td>
</tr>
</tbody>
</table>

Table 4.2: MaltParser and Parsey Universal LAS and UAS for all questions and embedded questions
4.3 Discussion

Generally, our hypotheses in section 3 were confirmed by our error analysis of the parsing output. The low accuracy for nominal modifiers suggests that this is indicative of parser error caused by dropped subjects and omitted prepositions. We further see that all models struggled with compounds, as we predicted, and that parser accuracy is lower for questions, which is consistent with our hypotheses. We also saw that higher root accuracy was tightly correlated to higher overall parser accuracy. We additionally observe that MaltParser’s edge over the SyntaxNet models (with gold POS tags) seems to stem from its ability to handle these types of constructions comparatively well, particularly compounds. We further suspect that because our data set contains a fewer number of long sentences and dependency relations, the fact that MaltParser’s performance degrades at a rate faster than neural network models for longer sentences and dependencies has limited effect on overall performance. In general, we have identified the areas that are challenging when parsing our data set, and in particular we identify questions, specifically embedded questions as a major challenge for parsing our data set. In the next chapter we discuss ways of mitigating these challenges to improve parser performance on our data.
5 Adapting the parser model

5.1 Building a better treebank

After our parsing experiments showed that parsing accuracy on our data set lagged several points behind than the English UD treebank, we wanted to test the extent to which several simple domain-adaptation techniques might close this gap. Because, as we discussed in section 3, the domain of chatbot dialogues has the potential to be quite varied, our goal was to find simple methods that do not require any human-annotated data, that can be implemented and adapted to specific dialogue systems should the need arise. To do this, we carried out several exploratory experiments, testing several different methods. Some of these methods, such as co-training, tri-training, and the use of a question-only treebank, QuestionBank, involved an attempt to generate in-domain annotated data for training. For the first two of these, we made use of 10,000 sentences of raw in-domain data. Additionally, we also investigated the role of sentence-length in training data. In our last experiment, we explore how pre-processing the input data can improve parser accuracy by eliminating certain problematic constructions.

5.1.1 Co-training

In order to capitalize on the respective advantages of both SyntaxNet and MaltParser, we tested using a combination of the two to annotate our 10,000 sentences of raw data. Because we observed that SyntaxNet was more accurate on data with predicted POS tags, we used Parsey Universal (trained on the EN-UD treebank) to produce POS and dependency annotations for these sentences. Because with accurate POS tags MaltParser had a 2 percent point accuracy advantage over Parsey Universal, we used MaltParser to train a model on the SyntaxNet-annotated data. We tested the usefulness of this data as a standalone treebank, as well as in ensembles with the UD English training data, and with QuestionBank (Judge et al., 2006).

5.1.2 Tri-training

We also investigated the effectiveness of tri-training, discussed in section 2.2.5 and in Weiss et al. For this method, we use Parsey Universal and MaltParser (both trained on the EN-UD treebank) to parse the 10,000 sentence raw data set, using Parsey Universal to generate the POS tags. We then select only the sentences on which both parsers agreed to train a third parsing model, in our case MaltParser. The data set this method resulted in was rather small, however the roughly 3,000 sentence annotations we obtained through this method are presumably of higher quality than those produced through co-training. Although the amount of data produced by tri-training could be
considered quite small for training data, we nonetheless use the output as a standalone corpus, as well as in conjunction with the UD English treebank and QuestionBank.

5.1.3 Using QuestionBank

We also experimented with the use of QuestionBank (Judge et al., 2006), which is a 4,000 sentence constituency treebank comprising only questions, which is intended to be a supplementary treebank to help parsers more accurately handle questions. We converted the treebank, which is annotated in the style of the Penn Treebank (Marcus et al., 1993) to Universal Dependencies using the treebank converter developed by Schuster and Manning (2016). Because our data contain a mix of questions and non-questions, we opted not to use QuestionBank as training data on its own, but used it to supplement our ensemble treebanks.

5.1.4 Results

To test the results of our co-trained, tri-trained and ensemble treebanks, we used each treebank to train MaltParser and tested each parsing model on the DTS with gold POS tags, as well as on the EN-UD test set for comparison. We tested the co-train and tri-train results as standalone treebanks, as well as each of these concatenated with the UD English training data, and in combination with the UD English training data added to QuestionBank. We also tested QuestionBank added to the UD training data without the supplementation of automatically annotated in-domain data. The results of these tests are listed in Table 5.1. While our best performing model did result in an LAS improvement of 0.3 of a percentage point, and a UAS improvement of half a percentage point, we saw similar improvements for the EN-UD test set, so it is difficult to say if this performance increase stems from the introduction of more in-domain data, or simply the introduction of more data in general. When we look at the results of the standalone co-train and tri-train treebanks, we do see some indication that the model is adapted to this domain, as both of these models perform much better on the DTS than on the EN-UD test set. The co-train treebank in particular scores quite highly, although fails to outperform the baseline model on the DTS. We also find the results of the standalone tri-train model to be quite encouraging, given that this model was able to achieve an LAS of 77.7 and a UAS of 83.5 with a treebank one quarter the size of the EN-UD training data. We are somewhat surprised to see that although adding QuestionBank to the UD training data lowers LAS and UAS on the DTS, when added to the co-train/UD-EN ensemble it outscores all other models.

<table>
<thead>
<tr>
<th></th>
<th>UD</th>
<th>CT</th>
<th>CT+UD</th>
<th>UD+QB</th>
<th>CT+UD+QB</th>
<th>TT</th>
<th>TT+UD</th>
<th>TT+UD+QB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DTS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAS</td>
<td>80.0</td>
<td>79.4</td>
<td>80.1</td>
<td>79.5</td>
<td><strong>80.3</strong></td>
<td>77.7</td>
<td>80.0</td>
<td>79.7</td>
</tr>
<tr>
<td>UAS</td>
<td>85.2</td>
<td>85.1</td>
<td>85.4</td>
<td>84.5</td>
<td><strong>85.7</strong></td>
<td>83.5</td>
<td>85.2</td>
<td>84.8</td>
</tr>
<tr>
<td><strong>EN-UD</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAS</td>
<td>84.4</td>
<td>77.0</td>
<td>84.6</td>
<td>84.4</td>
<td>84.5</td>
<td>74.9</td>
<td>84.4</td>
<td>84.4</td>
</tr>
</tbody>
</table>

Table 5.1: LAS and UAS for co-train (CT), tri-train (TT) and ensemble treebank models (QB refers to QuestionBank) compared to our baseline, trained on the EN-UD treebank (UD)
5.2 Selecting for sentence length

In this experiment, we explored whether altering standard training data to more closely reflect our own data set could improve parsing accuracy, in particular, given the significant difference in mean sentence length in our data compared to the UD-EN treebank, we sought to test if selecting similar length sentences to train on would benefit our parsing model. Our reasoning behind this was that shorter sentences would be more syntactically similar to those found in our data set. To do this, we selected sentences in the EN-UD training set under a certain sentence length, such that the average sentence length in our curated training data was 9.7, which is considerably closer to the 9.2 token sentence length found in our data set. This resulted in a training set of 7,564 sentences and 73,260 tokens. To control for training data size, we randomly selected sentences from the EN-UD training set such that our control treebank had roughly the same average sentence length as the full training set (16.9 tokens). This control treebank contained approximately the same number of tokens as our variable training set (78,719), but because of the difference in average sentence length, only 4,632 sentences. We then trained two MaltParser models using each of these treebanks. The results of this experiment are found in Table 5.2. As our results show, training on data containing only shorter sentences negatively affected parsing performance on both the DTS and the EN-UD treebank, although the negative effect was less pronounced on the DTS. This shows us that while training on only shorter sentences does not seriously hamper parsing on our data set, training on a mix of sentence lengths, including longer sentences, is beneficial to the parsing of our data set.

<table>
<thead>
<tr>
<th></th>
<th>M Sentence Length=9</th>
<th>M Sentence Length=16</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DTS</strong></td>
<td>LAS 78.8</td>
<td>79.1</td>
</tr>
<tr>
<td></td>
<td>UAS 83.9</td>
<td>84.6</td>
</tr>
<tr>
<td><strong>EN-UD</strong></td>
<td>LAS 78.5</td>
<td>80.2</td>
</tr>
<tr>
<td></td>
<td>UAS 82.6</td>
<td>83.7</td>
</tr>
</tbody>
</table>

Table 5.2: LAS and UAS on DTS and EN-UD test set with training data containing different avg sentence lengths

5.3 Question handling

In our final experiment, we investigated if our most successful adapted treebank had resulted in improved parsing performance on questions. We also test if manipulating the input data, namely, extracting embedded questions to stand on their own as sentences, made parsing easier. To do this, we manually extracted all embedded questions in the "Embedded Questions" subset of our data set, and re-annotated the dependencies in both the remainder of the sentence and the extracted question so that each part of the split sentence formed a complete dependency tree. For example, the embedded question described in section 3.2 resulted in the following two dependency graphs, as seen in figure 5.1. We then tested our most successful parser model from section 5.1, trained on the co-train/EN-UD/QuestionBank ensemble, on the questions, embedded questions, and extracted questions subsets of our data set, the results of
which we have listed in Table 5.3. We have included the results of the EN-UD model on these test sets for comparison.

Our results show that while our adapted treebank did not much improve parsing for questions in general (the 0.3 percentage point improvement is the same performance increase we saw on the entire data set), the LAS and UAS for embedded questions saw a substantial 4 percentage point increase, respectively. When parsing with the adapted parsing model, extracting embedded questions in the input data results in a further 3 percentage point increase in LAS, however UAS is not as drastically affected. For the standard EN-UD model, we see that parsing on the extracted question data set results in a considerable 6.9 percentage point increase in LAS over parsing on the embedded questions data set, with UAS actually higher than for the adapted model. This indicates that pre-processing the data to find and extract embedded questions could be a worthwhile pursuit.

<table>
<thead>
<tr>
<th></th>
<th>Full DTS</th>
<th>Questions</th>
<th>Embedded Questions</th>
<th>Extracted Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CT+UD+QB</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAS</td>
<td>80.0</td>
<td>79.7</td>
<td>70.6</td>
<td>73.6</td>
</tr>
<tr>
<td>UAS</td>
<td>85.2</td>
<td>85.2</td>
<td>81.2</td>
<td>81.8</td>
</tr>
<tr>
<td><strong>EN-UD</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAS</td>
<td>81.1</td>
<td>79.4</td>
<td>66.1</td>
<td>72.9</td>
</tr>
<tr>
<td>UAS</td>
<td>84.4</td>
<td>85.6</td>
<td>77.6</td>
<td>82.9</td>
</tr>
</tbody>
</table>

Table 5.3: LAS and UAS for questions, embedded questions, and extracted questions with standard and adapted training data compared to the full data set

5.4 Discussion

While the improvements we observed in the course of our domain adaptation experiments were modest, we believe that they are a promising start for further development. It would be interesting to try the same co-training and tri-training experiments, but with a highly accurate POS tagger trained and developed specifically for this domain. The cost of annotating POS data compared to full dependency treebanks is much smaller, and our earlier experiments have demonstrated the connection between POS accuracy and parser accuracy. With a more accurate POS tagger and a larger raw data set, we especially believe that tri-training could be used to construct a reasonably high quality treebank for this domain at relatively little human annotation cost. It would also be interesting to test the importance of quality over quantity in the data set. Here, we saw that 10,000 sentences of relatively poor annotation quality, but in-domain data could be used to train a parser nearly as accurate as one trained on 12,000 sentences of human-annotated data. We also saw that 3,000 sentences of shared parser output could train a model that performed not too far behind much larger, but either poor...
quality or out-of-domain treebanks. It would be interesting to investigate where the point of diminishing returns lies for adding more low-quality data vs. paring down the data to only high-quality parses. Our results for question parsing in our data set provide further room for future research, the first of which is the development of a method for reliable question extraction in dialogue data. This would likely not be too cumbersome a task and could likely be accomplished reasonably well through a rule-based method, as many embedded questions in our data set are tag questions (i.e. *I don’t need to pay a fine, do I?*), and the rest tend to follow somewhat predictable patterns. In general, our domain adaptation experiments show that while simple methods can make automatically annotated training data that is somewhat better suited to a specific domain, there is still no substitute for high quality human-annotated data.
6 Conclusion and future work

6.1 Using dependency-based syntactic annotation in practice

After evaluating dependency parsing on human-computer conversational data, and determining that we can achieve parsing accuracy nearly on par with standard English, the next step is to test the use of dependency-based syntactic information in the development of a dialogue system. We believe that there are several areas that have the potential to benefit from dependency parsing.

Studies have shown that dependency-based features can be useful for dialogue act modeling. Dialogue act modeling is the process of deciding whether a given input is a question, statement, suggestions, or an acknowledgement (Jurafsky and Martin, 2009), and constitutes an important part of many dialogue systems. In Boyer et al. (2010), the authors used a set of lexical, syntactic, and structural features together with a maximum likelihood classifier to assigns dialogue act labels to user utterances from a corpus of human-human tutorial dialogue. In doing so, they showed that lexical and syntactic features are highly useful for classifying dialogue acts in human-human dialogue in a task-oriented tutoring domain. We believe that this could be worth investigating for human-computer dialogue as well.

Finally, studies have shown dependency features to be useful in semantic frame parsing, discussed in section 2.6. For example, in Chen et al. (2016), it was shown that determining the most important substructures in a sentence by utilizing syntactic and semantic information improved both intention prediction and slot filling. It would be interesting to test if a similar methodology could be implementing in an existing commercial dialogue system.

6.2 Conclusion

In this thesis, we explored dependency parsing for human-dialogue system interactions, and sought to answer 3 questions, namely:

- What characteristics define dialogue data and what differences exist between dialogue data and standard English treebanks?

- How does overall parsing performance on dialogue data compare to standard English treebanks, and how do traditional transition-based parsers compare to state-of-the-art neural network parsers on our data set?
What steps can be taken to improve parsing on our data set, particularly with regard to automatically creating training data similar to this domain, and improving parsing for questions on our data set?

To address these questions, we built upon previous work in annotation, error analysis, and domain adaptation, and, in doing so, made the following contributions. First, in chapter 3, we annotated a data set consisting of human-dialogue system interactions according to the Universal Dependencies annotation scheme. We then carried out a statistical and linguistic analysis of the data, which to the best of our knowledge is the first such analysis of dialogue system data. We found that compared to a standard English treebank, our data contained a very high frequency of questions, in addition to having a much shorter average sentence length than our training data, which we hypothesized would negatively affect parsing quality. We further hypothesized that ambiguity caused by missing subjects, predicates, and prepositions, in addition to the high frequency of compounds, would also be sources of frequent parser error on our data set.

In chapter 4, our parsing experiments confirmed these hypotheses, showing that all parser models did indeed struggle with parsing compounds and questions. We also discussed the relative merits of a traditional dependency parser and neural network models, which we related back to our analysis of the data. In chapter 5, we tested several different methods of domain adaptation, which were informed by our data analysis as well as parsing experiments. We investigated the automatic annotation of in-domain data by leveraging the respective advantages of both MaltParser and SyntaxNet, showing that co-training using both types of models can result in a modest improvement over a baseline. We further showed that extracting embedded questions can improve overall accuracy for questions to a large degree. Our final contribution is an experiment showing the influence of training data sentence length on test data containing short sentences, in which we found that a model trained on longer sentences is more effective at parsing shorter sentences than a model trained only on short sentences. Having assessed how well dependency parsing works on human-computer conversational data, subsequent steps involve integrating a dependency parser into existing dialogue systems and finding ways to use dependency-based syntactic information to improve their performance and versatility.
Bibliography


