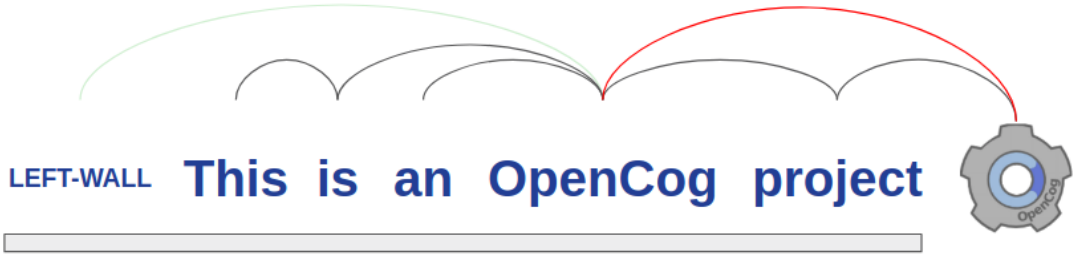
CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG



Statistical parsing and unambiguous word representation

in OpenCog's Unsupervised Language Learning project

Master's thesis in MPALG and MPCAS

CLAUDIA CASTILLO DOMENECH and ANDRES SUAREZ MADRIGAL

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Master's thesis 2018

Statistical parsing and unambiguous word representation

in OpenCog's Unsupervised Language Learning project

CLAUDIA CASTILLO DOMENECH
ANDRES SUAREZ MADRIGAL

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2018

Statistical parsing and unambiguous word representation
in OpenCog's Unsupervised Language Learning project
CLAUDIA CASTILLO DOMENECH
ANDRES SUAREZ MADRIGAL

Supervisor: Luis Nieto Piña, Faculty of Humanities
Advisor: Ben Goertzel, Hanson Robotics Ltd.
Examiner: Morteza Haghir Chehreghani, Data Science division

Master's Thesis 2018
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2018

Statistical parsing and unambiguous word representation
in OpenCog's Unsupervised Language Learning project
CLAUDIA CASTILLO DOMENECH and ANDRES SUAREZ MADRIGAL
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The work presented in the current thesis is an effort within the larger project entitled "Unsupervised Language Learning", aiming to build a system that can learn the grammar of a language by processing a corpus of unnanotated text. In particular, the authors focused on the first steps of the pipeline, including the selection and pre-processing of useful text collections to build and test its performance, as well as the syntactic learning loop in which the system obtains statistics from sentences in a given corpus and leverages them to implicitly learn the syntactic structure of the corpus language. The process gathers statistics from co-occurrence of words in a sentence using different counting methods and estimates the pair-wise mutual information between word pairs. Using this knowledge and using a minimum spanning tree algorithm, the system automatically produces syntactic parses of the ingested sentences. The unsupervised parses, when compared to human-generated or rule-based standards, show a varying quality; however, they always perform better than a baseline of randomly generated parses, implying that the system is indeed learning some of the assumed underlying linguistic content from the texts. From the results we learned that the complexity of the input text is a determining factor on the method that performs best, leading us to conclude that a successful unsupervised parser should be able to, up to some extent, pre-assess this complexity before processing. Also, the outputs of our different parser methods all show that accounting for distance among word pairs when parsing yields better results. Nonetheless, to get a more confident evaluation on this implication it is important to have a standard for comparison that bases itself on the same model assumptions. Additionally, we implemented a disambiguation process based in AdaGram as a way to build distinct representations for different word senses within a corpus, which then annotates the corpus with tags representing different uses of each word. The purpose of this pre-processing step is to break polysemy in the corpora and provide a cleaner input to the parsing pipeline. We report that our experiments show either slight improvement in parse quality, or no significant change, if disambiguated corpora are used as input.

Keywords: Unsupervised, language learning, parsing, disambiguation, computer science, engineering, project, thesis.

Acknowledgements

First of all, we want to thank the teams at Hanson Robotics and OpenCog for welcoming us in Hong Kong to contribute to their visionary goals and for sharing their knowledge, time, and friendship with us. In particular, we are grateful to Ben Goertzel for his insights and for making the collaboration between the university and the company viable, and to Anton Kolonin for his constant organization of our efforts and for being always ready to propose a new experiment.

Second, we want to recognize the contribution of our supervisor, Luis Nieto Piña, in making this thesis possible. We greatly benefited from his opportune and proper advice during the different stages of the project, as well as from his meticulous revisions of our manuscript, even during the rushed last days before submission. In all: ¡Gracias, Luis!

On the personal side, we have also received invaluable aid that we want to acknowledge:

Andrés: I am thankful for the unending support of my parents, brother and sister-in-law. I am also grateful to my friend Iuliia, who was always present during the good and less-good times of the past year.

Claudia: I want to thank my family for their unceasing support during my studies. I am especially grateful to my aunt María Sol, my brother Carlos and his wife Andrea, for not only hosting me in their homes, but also taking care of me, encouraging me and being my strength in the most critical moments.

Gothenburg, October 2018

Contents

List of Figures

List of Tables

1

Introduction

Language is perhaps the single most important feat that the human kind has achieved in history. Unparalleled by any other species on Earth, early human verbal communication offered a more efficient way to transmit ideas and abstract thoughts, and share experiences with others, thus accelerating learning and the progress of societies throughout the world. Current-day machines have their own internal languages, also invented by humans, and they're quite efficient with it. However, those differ considerably from natural languages (languages created by humans through evolution, without intentional design) to the point that only a small percentage of the population is fluent in them.

Natural Language Processing (NLP) is a multidisciplinary area of research that strives to create systems that are able to communicate externally through the use of human languages [29]. Using human language to express instructions to a computer and obtain its results would greatly enhance its efficiency of use. This is a non-trivial task, and although we already have decent speech recognition, sequential translation, and question-answering systems (just to mention a few applications), we still cannot make programs that fully comprehend language. Some even consider it an AI-complete problem: a problem that requires self-conscious systems, or 'true AI' to be solved [54].

Traditionally, approaches to tackle Natural Language Processing have depended on a human-in-the-loop, either to write sets of explicit rules that allow a system to disentangle the different parts of a language input, or to annotate text corpora for the system to learn from. The complexity of human language makes a set of rules or manual guidelines unlikely to cover all possibilities the language can offer, let alone when the semantic layer is to be taken into account, which seems necessary for a system to completely understand a language. Moreover, learning from text corpora in a supervised manner requires lots of data to be pre-processed and labeled before training, and unfortunately most of the annotating efforts have been centered in English (and a few ohter widely spoken languages), which means that these methods won't have quite the same efficiency in most other languages. Therefore, an unsupervised learning system that could derive meaning and learn any language's grammar (the structural rules present in that language) from unannotated corpora appears to be an adequate approach to solving the problem.

1.1 The Unsupervised Language Learning Project

The purpose of the Unsupervised Language Learning (ULL) Project[1], developed by researchers in Hanson Robotics[2] and the OpenCog Project[3], is to implement the NLP pipeline proposed by Vepstas and Goertzel [49], aiming to achieve the understanding of any language by training on unannotated text corpora. When completed, the process is expected to unsupervisedly learn a language grammar, a crucial tool needed for an NLP system to understand that language. The pipeline is based on a built-in linguistic model framework that supposes the existence of grammatical dependence, as well as syntactic and semantic structure in the language. The different layers of the model exploit and learn such components of the language. Figure 1.1 shows a high-level diagram of the described learning process.

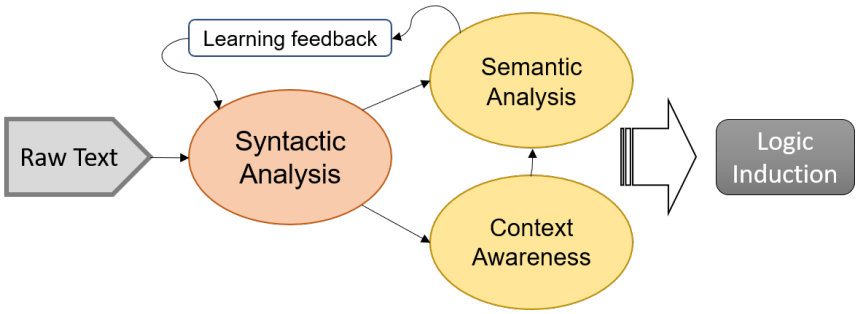


Figure 1.1: Language Learning Project pipeline.

In the diagram we can see how the units of the proposed pipeline interact in a recursive manner to learn the structures of each of the three language components, and finally obtain logic induction from them, all starting from raw text. In particular, the syntactic analysis processes receive feedback after the semantic analysis is performed. At the same time, learning the context of these structures can provide some tools for refined abstractions. The objective is for these components to be able to achieve a higher generalization of the structures present in the language. As the authors of the proposed pipeline state, "this can be viewed as a form of 'deep learning' at work: a higher, more abstract layer can serve to refine the correctness of a shallower, more concrete layer"[49].

After enough iterations, these three units of the language learning process should be capable of deriving "a system of relationships that display appropriate coherence and that, when applied by an appropriate parser to the inputs from the previous layer, will yield parse trees that reflect the information content of the input"[49]. By 'enough iterations' here it is meant a level of abstraction that will allow comparison and interaction with a model of the physical world so that the language model serves to represent it. Certainly, training all these in an unsupervised manner requires a large corpus of text, but once trained, the system will be able to understand

[1] `https://wiki.opencog.org/w/Language_learning`
[2] `www.hansonrobotics.com`
[3] `www.opencog.org`

previously unseen text by using the learned language structures and applying logic induction.

1.1.1 Unsupervised Syntactic Analysis

As one can see in the diagram (Figure 1.1), the first component of the learning process is the syntactic analysis. Although some implicit semantic analysis will be performed as well, the main focus of this thesis is centered on this first component; to highlight the difference we have used for it a different color from the other two components in the diagram. The next section discusses in detail the topic of this dissertation, but before getting into the specifics of it, we provide a short overview of the original authors' ideas about how to handle the extraction of syntax. In order to understand the particulars of the learning process, it is necessary to first understand the authors' general approach behind all of the component learning loops.

The high-level algorithm of the pipeline can be described by the following steps:

A. Define words to be 'objects'.
B. Look for correlations between objects.
C. Express the correlations in a convenient representation.
D. Cluster similar objects together into classes.
E. Define a new set of objects as the clusters obtained from the last step.
F. Repeat from B until certain level of abstraction is obtained.
G. Use definitions to apply logical induction on unseen data.

Observe that the 'feedback' to bootstrap the language components depends heavily on steps B, C, and D. In the context of syntactic analysis, finding correlations (step B) is exactly what a parser does. Thus, as the first steps in the ULL pipeline, the system should be able to observe sentences from an unannotated corpus and statistically induce a grammar from them, then obtain dependency parses with untagged labels between their components. These parses can be subsequently fed as input to the following learning layers, or continued to be processed to provide feedback for itself. In this way, recursion can be done at different levels by either using the feedback from the same component, or by taking in refinements from other components when redefining the 'new objects' accordingly.

Figure 1.2 shows how this recursion could be done in detail for the syntactic component. In a first pass of the currently proposed approach, raw input text is tokenized and separated into words, which will be our first 'objects'. Then, Mutual Information (MI) measurements are used to decide how likely it is for words to appear together in a sentence (as an induction of a rough initial grammar). Finally, a Minimum Spanning Tree (MST) algorithm uses this rough grammar based on word-pairs' MI to build a dependency parse tree for a sentence. In later passes of the algorithm, these same settings can be used to process the newly defined 'objects' [4]. Alternatively, we could use a grammar derived from our own method in further steps and

[4] For example, the new objects can be word-senses after disambiguation; or it can be words tagged with its categories obtained from a previous pass through the layer of semantic analysis.

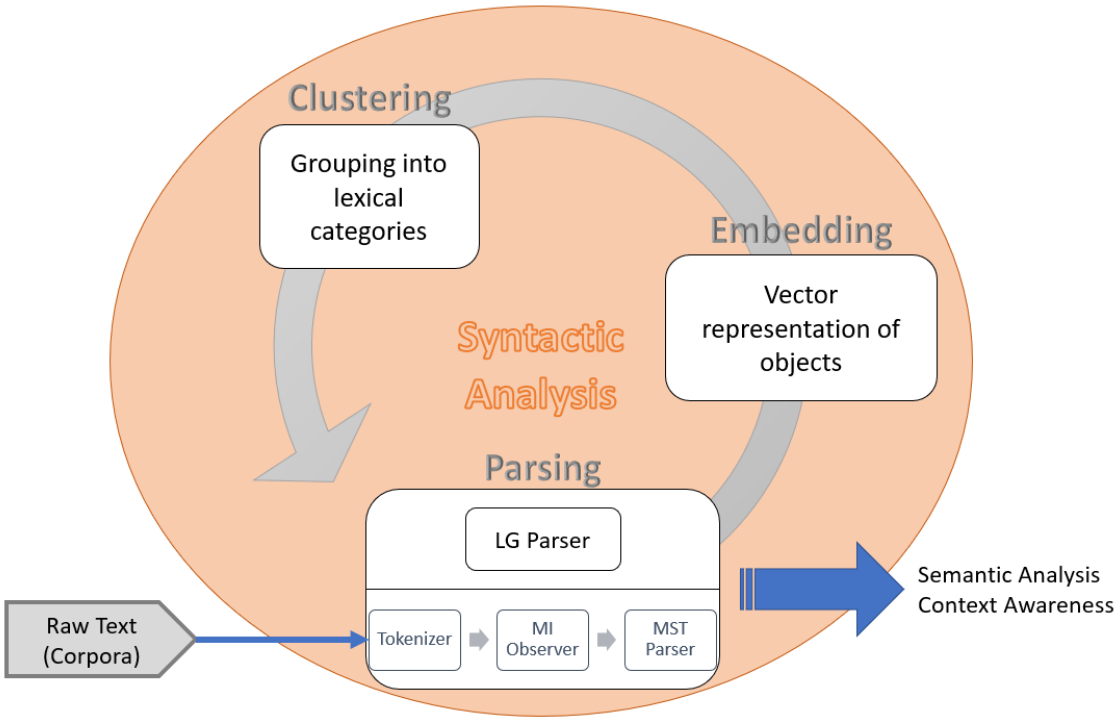


Figure 1.2: Syntactic Learning Loop.

parse the input in a dictionary-based manner.

Once the first steps are able to unsupervisedly produce acceptable parses from raw text, they will provide a database of dependent word pairs representing an enormous amount of highly fine-grained information regarding the syntactic rules of the language. The next natural step is organizing these relations to derive an improved structure of the learned language, and producing from it a proper grammar that could be comparable to hand-crafted rules like those used by, e.g. the Link Grammar [46][5]. This is equivalent to performing step D of the general algorithm; but in order to do that we first need to pre-process these parses or correlations in an appropriate way to allow for further organization and clusterization. In other words, we need to perform step C first.

A proper representation of our objects is therefore needed. Taking into consideration the information contained in the parses and the way most existing clusterization methods work, a vectorization of our objects seems appropriate. This is because having objects (words, word pairs or any other linguistic unit, in a first pass of the pipeline) represented in a geometrical space makes future comparisons and calculations simpler than having to deal with categorical objects. A possible approach is

[5] Link Grammar (LG) is a theory of syntax and morphology originally proposed by Davy Temperley and Daniel Sleator (1991), and contributed upon by several developers. It parses a sentence by directly connecting pairs of words instead of generating a phrase structure hierarchy like constituents grammars do. It differs from a dependency grammar in that links between words lack a head-dependent relationship, i.e. they are undirected. Further details as well as the parser code are available at the main website: `https://www.abisource.com/projects/link-grammar/`

to use dimensionality reduction based on context estimation: utilize a word embedding process that learns to represent concepts, in this case our objects, as vectors in a continuous multidimensional space[6]. This approach is convenient because most natural languages have in average more than 100,000 words in their vocabulary, and dealing with such high dimensionalities would be complex and time consuming, not to mention computationally expensive. Thus, using these smaller object vectors, it is possible to proceed with the next step: clustering of words into groups that resemble lexical categories.

The idea with clustering is to reduce the amount of highly fine-grained syntactic information obtained from the parser by extracting common patterns from it that can derive into a grammar. A suitable approach in early iterations would be to reduce the number of rules obtained from the parser by grouping together words that behave in a similar way, i.e. that frequently appear linked to the same word(s). The purpose of this categorization of words is to learn a grouping that could coarsely correspond to part-of-speech (POS) classes. In later iterations, when dealing with higher-level abstractions, the same grouping strategy could learn semantic relations that could be again fed back to the parsing and categorization system, and perhaps learn idioms and set phrases. Finally, the grammar rules obtained after clustering can be used to re-process the raw text, this time using a rule-based parser, or they can be fed as input to the following analysis component loops.

1.2 Contributions

The current thesis project strives to add to the first steps of this large NLP research effort by, as mentioned before, concentrating on the syntactic loop only. So far, we have presented a short summary of the proposal as it was originally stated by its authors [49][7]. It is important to note that the different pipeline components presented so far follow thoroughly-studied hypotheses that have been previously presented and partially tested by other authors, but had not been connected together to form a complete pipeline as attempted by the current project. Certainly, much has changed since the date of publication of the authors' first article, at the academic level and in relation to certain parts of the pipeline; and now we can use these advances to contribute to its development even further. Here we would like to broadly describe the actual implementations that our thesis project contributed with; but before we can do that, we need to provide a quick description of the development conditions of the general project before our participation.

When we joined the OpenCog project to work on the current thesis, there was already an alpha version of the parser with some experimental results, but it was not

[6] It is expected that this type of object representation will implicitly learn some meaning related to the objects. This is the only part of the syntactic loop that deals with semantics without receiving feedback from the semantic loop.

[7] For an alternative description of the project, check out the source code's Readme file at `https://github.com/opencog/opencog/tree/master/opencog/nlp/learn`

suitable for our purposes[8]. First, there was a description (Readme file and Wiki page) of the general project but no thorough documentation for each function and script existed, so every new person interested in collaborating with the project had to read through the entire source code to understand what each component did. Second, the installation process of the pipeline in a new computer was complex and there were some mistakes in the output of the parser in random runs that appeared to be related to this installation process. Third, the results available from experiments conducted with this alpha version of the parser were not properly organized, it was unclear which specific corpora had been used as input in each case. Also, at least some of the results were obtained using Wikipedia articles as the input text, which did not provide good results given that the fact-informative nature of these articles bring little variety in pronouns or action verbs, and contains large numbers of proper names, foreign language words, dates, tables, etc., that hinder the learning process and can lead to unusual deductions of grammar.

Given the development state of the project, it seemed like a logical way to contribute to start by gathering useful corpora, implementing a pre-processor of data, revising and amending the state of the alpha parser, and possibly working with word sense disambiguation before parsing as part of the syntactic analysis loop. With this in mind, we identified the need for five specific tasks to be developed:

1. Find or create adequate text corpora to test the efforts of the project.
2. Create a configurable pre-processor to simplify these corpora as needed.
3. Understand, modify and test the existing OpenCog-based parser to make it useful to the project.
4. Integrate word disambiguation and representation learning algorithms (as part of pre-processing).
5. Write proper documentation for the three previous tasks.

In particular, the question we aim to answer in this thesis project, by performing the described tasks, is whether it is possible to learn the syntactic structures of a language in an unsupervised manner and get results that are at least as good as the existing supervised methods. We will try to achieve this by assuming that each language has an innate dependency structure between word pairs and aim to learn this structure from observing a lot of text. As part of the ULL project, our ideas will be based on the authors original proposal, but we will use our knowledge of the latest developments in the field to try to advance even further. Technically speaking, the goals of this thesis project are to improve, combine and modify existing algorithms (which may be used for other purposes) to work as a part of the proposed pipeline. According to the results some extensions to this project are proposed in the Conclusion chapter. Finally, one challenge we aim for is making these modifications using graphs and tree-like structures in such a way that the resulting algorithm can be applied to numerous other fields outside NLP.

[8] One of the project contributors experimented with the other two parts of the syntactic loop as well, but none of his results at that moment looked like something we could build upon.

1.3 Scope and Limitations

This thesis project deals with a problem that is part of a bigger and rather ambitious research effort: solving unsupervised language understanding. If such a research effort succeeded, the conceptual contributions to the scientific community would be immensely valuable. Nowadays, developing an unsupervised language parser for any language would already be a big milestone. We considered that solving the problem of obtaining simple grammar and syntax structures and finding a proper representation for them to improve a posterior grouping into lexical categories, is a key piece necessary before experimenting with recursion. Achievement of favorable results in this topic would be equivalent to overcoming the next obstacle, allowing for further development in more abstract layers of language abstraction and understanding. Thus, this is the current focus of the OpenCog team: experimenting with one forward pass through the syntactic loop only.

Moreover, as part of the OpenCog team research project, the results of this thesis will be limited by one pass of the pipeline as well. It is important to keep in mind that according to the authors' proposal, refinements in learning are obtained through recursion, so without recursion the outcome of a first pass through the pipeline is expected to be of limited performance. For this reason, we will also limit our evaluation standards, with the hope that once recursion is implemented in the future, our method will succeed to surpass not only the state of the art of unsupervised methods but supervised methods as well. It is also relevant to mention that the scope of our thesis does not involve the embedding nor clustering sections[9]. Therefore, we found the experimentation with word-sense disambiguation before parsing to be a convenient way of evaluating how the parsing outputs can vary if the input to the pipeline (our 'objects') get refined.

Finally, because we won't be able to harness the power of recursion during a single pass of the learning algorithm, we decided to simplify the input data through a pre-processing step, in order to understand better if the implemented ideas work in a simple case, thus assessing their performance. Because of time restrictions, we decided to focus on texts that have different complexity levels but that belong to the same language. Since the most extensive supervised NLP efforts have been realized in English, we restricted our experiments to English corpora as well, so that we have a benchmark for comparison[10].

Following the previous explanations, the structure of this paper is as follows:

- Chapter 2 introduces the related work relevant to our topic. It specifically ex-

[9] Other members of the OpenCog team are working in parallel to us with these sections. Unfortunately, because of time limitations their results will not be possible to publish or use for comparison within this thesis.

[10] One of our collaborators has tested the pipeline with Chinese, but his scripts require the text corpora to be first processed by another algorithm that learns to split the text into words before inputting to the pipeline, and this effort is outside of our thesis domain.

plains the two main theories that our thesis will develop on, which are Yuret's unsupervised parser [55] and AdaGram word embedding [2]. It also gives an overview of the OpenCog project.

- Chapter 3 presents the ways we will implement the ideas taken from the related work in the OpenCog environment and what tools we will use to evaluate our results. This chapter offers also a description of the text corpora we selected.

- Chapter 4 presents the results obtained from our experiments, mainly comparison charts and useful output examples.

- Chapter 5 summarizes our findings, gives recommendations of what could be improved and suggests ways in which this project could be continued in the future.

2

Related Work

The amount of research involving methods for natural language processing systems is extensive. Nonetheless, as mentioned before, most of the existing approaches have a high dependency on the input of human experts on the field. Klein and Manning [31], more than a decade ago, summarized results from efforts towards learning a grammar without human supervision, while exposing the difficulties with such a task. When it comes to unsupervised learning of specific tasks (for instance syntax parsing, word sense induction, or document translation), some attempts have been made, but none of them can equal the results obtained by their supervised counterparts, which most of the time are limited to one language, or even one context. A more recent appraisal of the general task of language acquisition is offered by Dupoux [19], where stress is made about using, at most, a limited amount of supervision and 'wild' data, similar to the one that children are exposed to.

In this context of new approaches, several unsupervised parsing algorithms that make use of Harris' distributional hypothesis [24] are available in the literature [31][15][11], which leverage some linguistic knowledge (e.g. part-of-speech of words) and are therefore not entirely unsupervised. To our knowledge, the state of unsupervised parsing has not improved considerably in the last decade. Deniz Yuret [55] built a truly unsupervised parser based in maximization of mutual information between words, which did not perform as well as supervised approaches, but he pointed out potential improvements. With respect to word embeddings, methods to unsupervisedly learn dimensionally-reduced real-valued vector representations of words have been proposed in the past [5] [36, 37] [39]. Algorithms like these would prove useful for our current ULL purposes, as a way to learn representations for words that would allow for word-sense disambiguation and refinement of the input data.

The iterative Unsupervised Language Learning (ULL) pipeline originally introduced by Vepstas and Goertzel [49] brings together generalizations and extensions of previous theories by these and other authors to tackle language learning. A complete description of such theories is respectively cited in their article, but for the purposes of this project, we will make a short review of the most relevant ones. Our work makes use of algorithms for parsing and for word representation learning in an unsupervised manner. This chapter presents the theory behind those algorithms.

2.1 Language models

If we want a computer to 'understand' a human language, we first need to find an adequate way to represent it. When it comes to modeling, a natural question that arises is if a model can actually exist within such a complex process. The idea that a distributional structure is present in both the written evidence of the language and the speaker itself is old and well studied [24]. For example, several experiments show that humans have different levels of associations between diverse types of words, which can be measured by comparisons in their response times after exposure to them [14]. Hence, the question is reduced to: which is the best model given a particular context or purpose?

On the pursuit to answer this question, context-free grammars have historically been used to describe the structure of sentences and words in a natural language. Their name derives from the fact that it uses simple replacement rules which are always applicable, independently of the context in which they are found. A lexical system is a language model that describes the syntax or structure of the language in terms of its basic primitives, like words or phonemes. Most of the lexical systems, and in particular the ones upon which this project develops, are based on context-free grammars.

The first lexical systems using probability to mathematically represent the structures present in language were the n-gram models [29], where an expectation value was assigned to a word given the previous n-words in the sequence. These models report fair performances when applied to particular contexts, but are mostly limited to simple sentences. If we want a model to capture more complex relations, such as the ones determined by compound sentences, or word sequences longer than the n-window scope of n-grams, we need a different kind of model. Bengio *et al.* [5], argue that one needs to learn the joint probability of the words in the sequence for that purpose, and thus face the curse of dimensionality[1]. They introduce a neural probabilistic model to solve this.

An alternative to the massive use of computational power that complex modeling or neural networks require, is the use of independence assumptions. A dependency grammar, for instance, bases its analysis on the direct relation between the linguistic units, e.g. words, and assumes that more complex relations can be built upon these. A parser that is based on a dependency grammar tries to assign a dependency structure to every sentence. A dependency structure is a tree (in the graph sense) composed of planar directed arcs that connect all the words in a sentence, in such a way that each arc connects only two words at a time and no arc passes over the root word [47]. These characteristics of dependency structures make it easier for the parser to gather statistical information about the distribution of words relative to each other.

[1] This expression was introduced by Richard E. Bellman within the field of dynamic optimization [4]. It the context of machine learning it refers to various problems, such as data sparsity or computational deficiency, that derives from trying to analyze data in high-dimensional spaces.

On the late 90's, Yuret introduced a formalism based on dependency structures, where the relation between individual pairs of words are the basic primitives of the language [55]. He argued that basing the analysis of a language only on a fixed amount of rules does not suffice because both syntax and semantics are needed to fully understand the language, and that both complement each other. In his words, "what makes humans good learners is not sophisticated learning algorithms but having the right representations" [55]. In his PhD thesis he refers to studies showing that children automatically learn the likelihood of the connection between concepts from their experience with the world, and use this knowledge to break the ambiguity in meaning, even without explicitly knowing any syntax rules. Moreover, he sustains his posture that syntax alone is not enough given the single-dimensionality of language: only a finite (limited) amount of different arrangements of words are possible, giving rise to the one-to-many mapping between conceptual relations (meanings) and syntactic relation (ordering of words).

2.1.1 Lexical attraction

For the purpose of this thesis, we chose to model language the same way Yuret did when he presented his first unsupervised parser based on lexical attraction. Lexical attraction models are a class of probabilistic language models founded on the aforementioned formalism where the language's 'building blocks' are the words. Lexical attraction is defined as the measure of affinity between those building blocks, i.e. "the likelihood that two words will be related in a given sentence" [55]. The concept of lexical attraction is based on the information-theory concepts of entropy and mutual information.

2.1.1.1 Entropy

In his Mathematical Theory of Communication, Shannon [45] proposes the idea of understanding words in a sentence in a natural language as a sequence of tokens which can be represented by a discrete random variable. He defines the entropy of such variable as the average information content that each token in the language can have. Mathematically entropy is calculated by the formula:

$$H = -\sum p_i \log(p_i)$$

where i ranges over the possible values of the random variable and p_i is the probability of the variable taking the value i. Note that it is a weighted average of the 'information content' of each possible event, which in turn is a negative logarithmic function ($-\log(p_i)$). Consequently, events that are less 'likely' to happen (i.e. have a low p_i) are considered to have more information content because they are less expected.

2.1.1.2 Mutual Information

Mutual information (MI) is a measure of how much more information content can be extracted from a random variable, given the knowledge of another random variable

[17]. It is calculated as [2]:

$$I(X,Y) = \sum_{x \in X, y \in Y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)}$$

Like entropy, mutual information is also a weighted average of a logarithmic function of probabilities, but in this case the function has two random variables. Thus, mutual information describes the dependency between those two variables. MI can actually be expressed in terms of entropy:

$$I(X,Y) = H(X) - H(X|Y) = H(Y) - H(Y|X) = H(X) + H(Y) - H(X,Y)$$

These identities provide a more intuitive and Bayesian way of understanding MI. In particular, it allows us to see that MI is symmetric and non-negative, and that it is equal to zero if and only if the two random variables, X and Y, are independent of each other.

Yuret defined lexical attraction as the likelihood of a syntactic relation [55]. MI compares probabilities and is therefore a measurement of likelihood. In the equation above, if we take X to be the random variable related to one word appearing in a certain position of a sentence and Y to be the same but for another word and position in the sentence, then the mutual information between the two variables is a measure of how much information content do the two words appearing in particular order and place provide to the sentence structure. This particular order and place is exactly what a syntactic relation encloses. In this sense, lexical attraction can be understood as the mutual information captured in the syntactic relation between X and Y.

Moreover, MI measures the average entropy between all the possible outcomes of two random variables. However, in some cases we might be interested in quantifying the information content of an individual pair of events. For example, if we want to know what is the likelihood of two specific words in a vocabulary being related. In such cases we are interested in what is called the point-wise mutual information (PMI), also known as fractional mutual information (FMI) [14]:

$$I(x,y) = \log \frac{p(x,y)}{p(x)p(y)}$$

Note that $I(x,y)$ here is short for $I(X = x, Y = y)$.

Unlike MI, PMI can be negative ($I(x,y) < 0$) and this happens when x and y are in complementary distributions (for example, antonymous words) because $p(x,y)$ is

[2] It is important to comment about the notation used here. In this equation $p(x)$, $p(y)$, and $p(x,y)$ are short for $p(X = x)$, $p(Y = y)$, and $p(X = x, Y = y)$ respectively; where X and Y are different random variables with possibly different sample spaces. This means that even though by symmetry $p(X = x, Y = y) = p(Y = y, X = x)$, it is not necessarily true that $p(x,y) = p(y,x)$. The reason is that, to be consistent $p(y,x)$ in this context is short for $p(X = y, Y = x)$ and not for $p(Y = y, X = x)$; the first of which may not even be admissible.

very small compared to $p(x)p(y)$). PMI can also be approximately zero ($I(x, y) \sim 0$) if there is no interesting relationship between the two events. In contrast, when there exists a high association between the events x and y, their joint probability $p(x, y)$ will be much larger than observing them by chance or individually ($p(x)p(y)$); hence their PMI will be positive ($I(x, y) >> 0$). Accordingly, in the previous example, word pairs that appear frequently together will most likely have high PMI.

2.1.2 Maximal parsing algorithm

Yuret's model of lexical attraction assumes that words with high lexical attraction are likely to be syntactically related within a language. Taking the information-theory approach, the probability of observing a sentence can be computed as the product of the probabilities of observing each word (e.g. an unigram model); but under the assumption of a dependency grammar model, words are not independent of each other. Thus the probability of a sentence depends as well on the conditional probabilities of the relationship between the words in that sentence. To avoid the complexity of long term dependencies, Yuret shapes his lexical attraction model as a Markov network, where each word depends on only one other word in the sentence, not necessarily an adjacent one, but it's independent of all the others. He then shows that the total entropy of a sentence in such a model is completely determined by the lexical attraction between the dependent words. In this way, because the most probable language model is also the one that produces the lowest entropy, he proves that "the search for a low entropy language model leads to the unsupervised discovery of syntactic relations" [55].

Furthermore, upon this discovery he tackles language understanding through the use of an algorithm for maximal parsing. The goal of the algorithm is to find the dependency structure that assigns a given sentence a high probability based on the lexical attraction. In other words, the algorithm's objective is to find the parsing tree (which by definition has no cycles and a planar structure) which assigns the highest total mutual information to the words linked by the tree. He describes an algorithm based on Viterbi that obtains the optimal solution in $O(n^5)$ time, where n is the number of words in a sentence. He also proposes an approximation of his own which uses the properties of planarity and cycle restriction and has a performance of $O(n^2)$. This approximation algorithm does not always find the most likely parsing tree and in some cases it might leave some words disconnected, but in practice it has a good average performance and its simplicity in implementation makes it a good candidate for training.

2.2 AdaGram: learning disambiguated word representations

The impact caused by the deep learning and neural network revolution has reached language learning, as well. By early 2000's, Bengio *et al.* [5] introduced a system

that learned distributed representations[3] for words in an unsupervised way by using their context. Inspired by statistical language models and the distributional hypothesis [24], they obtained better results than the best existing n-gram models of the time, even more efficient than Latent Semantic Analysis [18] and Latent Dirichlet Allocation [8]. Their proposed neural network was able to learn both word representations and a language model at the same time.

Improvements followed in the coming years. Particularly, Mikolov and colleagues' Skip-Gram model [37, 36] grabbed the community's attention because of its ability to predict the context of a word with high computational efficiency, making it suitable for online processing of text. Skip-Gram stood out as well for the unexpected semantic content embedded in the learned vectors: some algebraic operations on word vectors give results similar to analogy tasks. For example, if one subtracts the vector for the word 'man' from the one for 'king', and add the vector for 'woman', the result is very close to the vector for 'queen'.

A problem not addressed by Skip-Gram is that of ambiguous words. When a word has different meanings that occur in different contexts, it has to average between its learned representations and ends up dominated by the most common use of the word. Hence, in the best case, a word will be correctly predicted from its dominating context and ignored in the other ones. In a less fortunate situation, the resulting embedding will not be properly predicted for any of the word sense's contexts.

AdaGram (Adaptive Skip-gram) [2] is a neural network-based continuous representation learning algorithm based in the Skip-gram model which addresses the aforementioned limitation. It is modified to build a different vector representation for each of the significant word's senses that appear in the training data, while still offering speed of processing and meaningful representations. Various other neural network architectures have been proposed to accommodate ambiguity, but they lack in processing efficiency [27], require external knowledge like WordNet senses [13], or allow only a limited/fixed number of senses per word [41, 48].

As part of the ULL pipeline, AdaGram[4] will be used for two different purposes: to disambiguate polysemous words, and to learn word representations that can subsequently be used for finding syntactic categories in the language. In this thesis we will tackle only the first purpose. AdaGram takes unlabeled text as input and learns different representations for each word in the vocabulary by training the network using word pairs taken from the context of the target word, following the principles of the distributional hypothesis [24]. Our expectations for this thesis is that the

[3] A distributed representation is a way of expressing an entity (in this case a word) with "a pattern of activity distributed over many computing elements", where each computing element is involved in the representation of other entities as well [26]. When it comes to word embeddings, computing elements are associated with the dimensions of the vector space, where each dimension is related to a characteristic of the word (part of speech for instance). Distributed representations are known for its computational efficiency because of its network structure with simple, neuron-like computing elements.

[4] The repository containing AdaGram can be found in `https://github.com/sbos/AdaGram.jl`

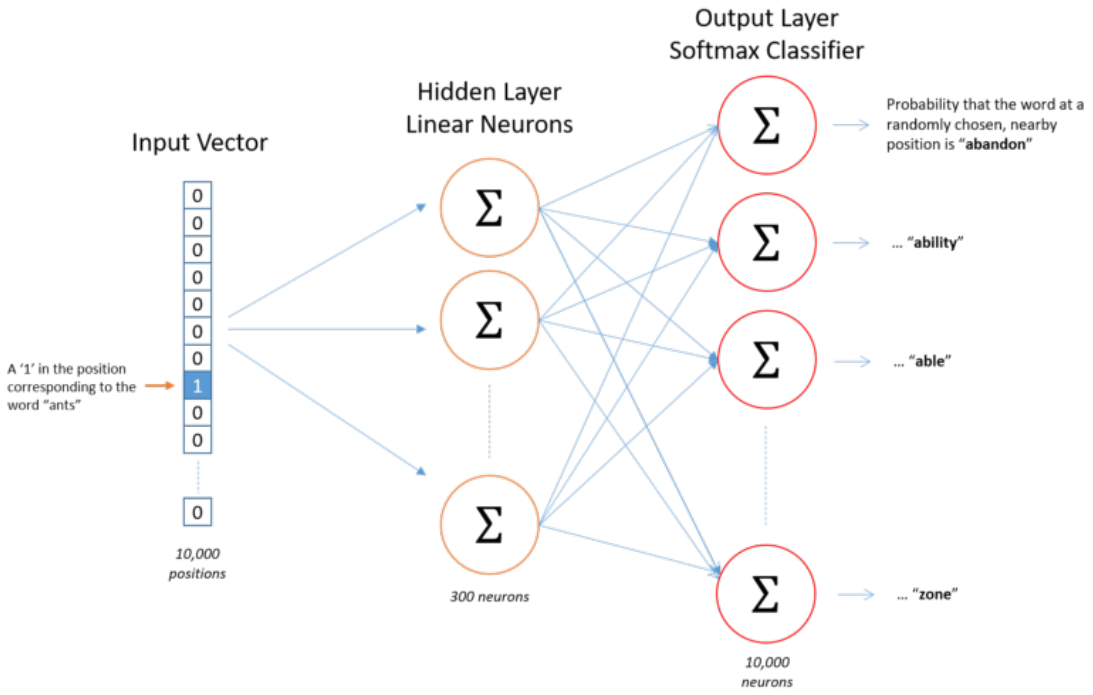


Figure 2.1: Architecture of the Skip-Gram model. Image credit: Leonardo Barazza [1].

learned representations correspond to useful word senses, but in order to understand how this can happen we need to go through the algorithm. First, we'll introduce how Skip-Gram works, and then the modifications included in AdaGram to offer multiple prototypes per word.

2.2.1 Word embeddings in Skip-Gram

A great deal of the success of Skip-Gram comes from the simplicity of the network used to build word prototypes (as embeddings are regularly called), which allows for a low complexity during training and operation. The full details of the network's operations can be found in the given reference, but its architecture is shown in an example in figure 2.1. The network is composed of a one-hot input layer of dimensionality V that codifies the input word as an index, a fully-connected linear hidden layer of size D, and a hierarchical soft-max output layer with V neurons. V here corresponds to the size of the training vocabulary (10000 in the example) and D is a hyper-parameter of the model (300 in the example) which determines the reduced dimension (i.e. $V >> D$) of the learned representations.

For training, each word x_i of the input text X (containing N words) is assigned a context $\mathbf{y}_i$ from its C neighboring words (another hyper-parameter) both preceding and following it, all converted to an index in the range $[1,V]$ using a simple look-up table built from the whole corpus. Then, the currently trained word is fed into the network in one-hot encoding format, and one of the context words is used as the training sample to evaluate the loss function; error backpropagation follows. This

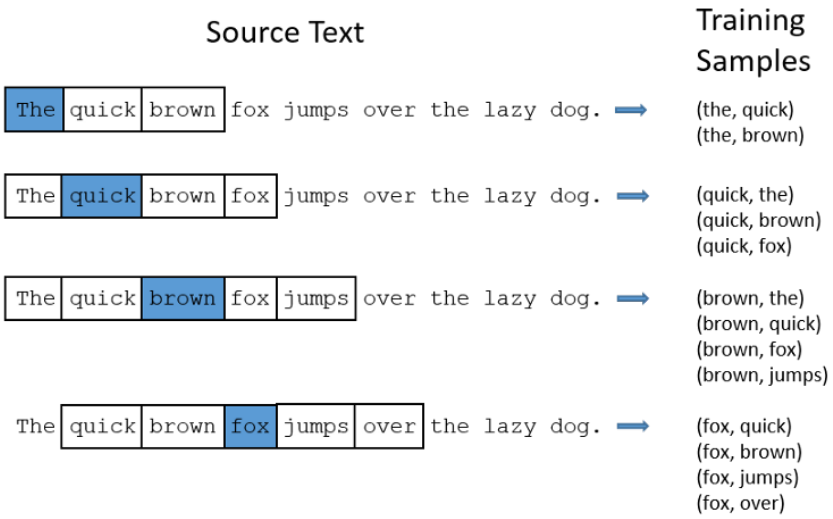


Figure 2.2: Training sample selection in the Skip-Gram model. Image credit: Leonardo Barazza [1].

task is repeated for the rest of the context of this word, and the whole operation for every word in the corpus. This constitutes one training epoch. Figure 2.2 illustrates the training sample selection in the case of a context of size 4, where x_i is shown in blue, and $\mathbf{y}_i$ is represented by the white rectangles. Notice how the size of the context is smaller for words near the limits of the text.

As stated by Bartunov and colleagues [2], Skip-Gram's training objective is to find the parameters θ (the neural network's weights) that maximize the likelihood of context words Y, given the training words X. Reproducing here their equation 2, that is:

$$p(Y|X,\theta) = \prod_{i=i}^{N} \prod_{j=1}^{C} p(y_{ij}|x_i,\theta).$$

The loss function is simply the mean squared difference between the hierarchical soft-max values and the one-hot vector representing the expected context word at each training step. Once the network is trained, Mikolov *et al.* use the parameters between the input and hidden layers as continuous vectors to represent each of the words.

2.2.2 Multi-prototype AdaGram

The Adaptive Skip-Gram model of Bartunov *et al.* introduces modifications to the Skip-Gram architecture, aimed to allow a single word to potentially have multiple embeddings that refer to its use in different contexts, or senses, thus offering a solution to polysemy. These modifications basically amount to introducing T input neurons per word (a hyper-parameter which specifies the maximum number of meanings a word can have), with their respective connections to the unchanged hidden layer. The output layer is not modified and holds only one neuron per word in the

vocabulary, thus being trained on word samples given sense inputs.

Because not every word is expected to have the same number of senses, AdaGram uses a Dirichlet Process allocation of new senses that 'activates' new prototypes based on a prior distribution dictated in the stick-breaking style of Sethuraman [44]. With these modifications, the AdaGram model is described by Bartnuov and colleagues' [2] equation:

$$p(Y, Z, \beta|X, \alpha, \theta) = \prod_{w=1}^{V} \prod_{k=1}^{\infty} p(\beta_{wk}|\alpha) \prod_{i=1}^{N} \left[p(z_i|x_i, \beta) \prod_{j=1}^{C} p(y_{ij}|z_i, x_i, \theta) \right],$$

where α is an an extra hyper-parameter that, together with the frequency of a word, influences the number of senses that word will have; $Z = \{z_i\}_{i=1}^{N}$ represents the senses for all words; and β is a Dirichlet Process parameter. Accordingly, the loss function and training updates are modified to accommodate these changes.

The reader is advised to review the original article for full details or an in depth explanation of the network design. It is relevant for us to mention though, that the training algorithm's complexity scales only linearly with the hyper-parameter T Because T represents the maximum number of possible representations for each word, usual values of T are fairly small so the algorithm maintains its efficiency.

2.3 OpenCog: an open source platform for AGI

Inspired by the architecture and dynamics of a human brain, OpenCog is an open source software framework that offers a diverse collection of cognitive algorithms [38]. Ranging from perception and learning to creativity and socialization or memory and language, among others, the platform hosts a carefully selected combination of algorithms that tackle different kinds of knowledge. The objective is to imitate the environment and development of young human children by integrating tasks of different complexity involved in all major aspects of human intelligence. Most of its source code is written in C++, but in the attempt to make it scalable and robust, it offers withal support for Scheme (guile) and Python bindings.

The mission of the OpenCog project is to achieve Artificial General Intelligence (AGI); that is, developers of the project aim to mimic with computers the general intelligence at the human level, and hope to one day go beyond it. In a shorter term, the goal is to provide to the technological community with "code that can be used (piece by piece or as a whole) to help make practical software applications smarter" [38]. Even though the project is at an early stage of development, its lead researchers[5] have a long trajectory in the AI field. Ben Goertzel[6], founder and leader of the OpenCog project, has published a number of books and articles about AGI

[5] `https://opencog.org/leadership/`
[6] `https://en.wikipedia.org/wiki/Ben_Goertzel`

(term which he coined) [22, 52, 21, 23], leads the AI efforts in a handful of companies[7], and organizes a yearly Conference on AGI[8]. What's more, some of OpenCog's most robust algorithms are nowadays in use by more than 50 global companies, including Huawei and Cisco [42].

At its core, the OpenCog project consists of a knowledge representation database, the **AtomSpace**, and a series of assorted projects which are under development. These last include, but are not limited to, natural language processing, attention allocation, spatial and time data managers, embodiment and psychological states modeling, and network server scripts (the cogserver)[9]. The AtomSpace provides these projects with the work-space and tools necessary for learning and reasoning algorithms to be developed. It comprises an in-RAM database and the associated query/reasoning engine to fetch and manipulate the data on it. Data in the AtomSpace is represented in the form of hyper-graphs[10]; ergo, the AtomSpace can be seen as a type of graph database in which the query engine is in charge of 're-writing' the graph system and the rule-engine undertakes the general 'rule-driven inference'.

The vertices and edges of such hyper-graphs are called **Atoms** (hence the name of the database), and can be used to represent not only data information but also procedures. Consequently, many of these hyper-graphs represent executable programs as well as data structures. The 'programming language' created to manipulate the atoms and construct the corresponding hyper-graphs is called the **Atomese**. This language resembles a mixture between several programming languages such as SQL (for query), Prolog (for logic), Lisp (for lambda expressions), and others, because of their utilities. It was not meant to be used by humans, but it was rather designed for machine automation.

Atoms can be understood as type instances in type theory or as symbols in many programming languages. They are permanent and immutable, but they can have a value or an interpretation attached to it, which in turn can be mutable. Multiple, different values can be associated to a single atom, and each of them can be accessed with a unique key. The triplet (atom, key, value) is called a *Valuation*. Valuations normally have fleeting, changing values that indicate the truth or likelihood of the atom they are attached to, but they can also contain other kinds of transient data. There are many different atom types as well, some used for basic knowledge-representation and computer-science concepts. For example there exist atoms representing relations, such as similarity, inheritance and subsets; atoms used for logic, such as Boolean and, or, for-all, there-exists; along with many others.

[7] Like singularitynet.io, hansonrobotics.com, mozi.ai, among others.

[8] `http://agi-conference.org/`

[9] Source code at `https://github.com/singnet/opencog/`

[10] A hyper-graph is a generalization of a graph where each edge can connect more than two vertices at a time.

2.3.1 Representation structures

In the context of this thesis there are two atom types that are of particular interest: Nodes and Links. These two types of atoms give the basis for all of the representation structures that we use for statistical gathering of language text. Definitions follow, along with pieces of code that help to clarify concepts for curious programmer-readers.

Nodes contain a name or label and a type, which cannot be changed after creation. These two properties are the only ones they have and they form their identification key. As any other atom, once a node is placed in the AtomSpace it is unique: there can only ever be one node with a given (name, type) pair. If a second Node with the same key was to be inserted in the AtomSpace, any valuations associated with it would be merged with the ones from the previously inserted Node. The following piece of code shows the class definition in C++ for a Node atom.

```
class Node : public Atom
{
    private:
        std::string name;
    public:
        Node(Type, std::string &);
        const std::string& getName() const;
        std::string toString() const;
};
```

Links are used to connect or associate together other atoms. Unlike nodes, links do not have any name or label; they are identified uniquely by their type and contents. The content of a link is a set of other atoms, which can be ordered or unordered, and is called the link's *outgoing set*. The C++ class definition for a Link atom is shown in the following piece of code:

```
class Link : public Atom
{
    private:
        std::vector<Handle> _outgoing;
    public:
        Link(Type, const std::vector<Handle>&);

        const std::vector<Handle>& getOutgoingSet()
            { return _outgoing; }
        size_t getArity() const {return _outgoing.size(); }
        std::string toString() const;
        std::string toShortString() const;
};
```

2.3.1.1 Sentence representation

In order to process natural language using the AtomSpace, it is necessary to have ways of representing the sentence structure of a document using Atoms. In Opencog, the sentence representation follows a hierarchical order, with sentences grouped (through links) into documents, parses grouped into sentences, and word instances grouped into parses[11]. There are also other association structures such as word instances corresponding to a word node, or linkage between different word instances (syntactic links), as explained with greater detail in section 3.2.

The notation used to define Atoms is the one for constructing them using Scheme language. In a generic way it is as follows:

```
(Type "identifier@UUID" TV)
```

where the $Type$ is the atom type, the $identifier$ is a unique string which serves also as a key handle, $UUID$ is a 128-bit MD5 hash printed in ASCII to make it readable, and TV is a truth value of the atom (a valuation as previously defined). Next, we will introduce the most relevant sentence structures that will be used for statistical gathering within our thesis.

- For documents:

  Multiple sentences within a document are connected using a **SentenceLink**. Each document is saved in a **DocumentNode** and every sentence is saved in a **SentenceNode**. There is only one DocumentNode per document and one SentenceNode per sentence.

  ```
  (SentenceLink
      (SentenceNode "sentence@UUID")
      (DocumentNode "document@UUID")
  )
  ```

  This type of link only indicates if a given sentence is a part of the document but it doesn't indicate where it is located within the document. A **SentenceSequenceLink** indicates sentence order with a NumberNode. Note that '42' is just an example here.

  ```
  (SentenceSequenceLink (stv 1.0 1.0)
      (SentenceNode "sentence@UUID")
      (NumberNode "42")
  )
  ```

- For sentences:

  A **ParseLink** connects parses to their original sentences. A parse is saved in a **ParseNode** which has a SimpleTruthValue associated to it. SimpleTruth-

[11] Documentation available at `https://wiki.opencog.org/w/Sentence_representation`

Values provide a ranking for such parses as a numerical confidence value v. This is grant because some of the parsers that OpenCog supports can output several parses per sentence, each one with an evaluation, which can be saved as such confidence value.

```
(ParseLink
    (ParseNode "sentence@UUID" (stv 1.0 v))
    (SentenceNode "sentence@UUID")
)
```

- For words:

  Each word in a vocabulary is represented with a **WordNode**, where the identifier is the word itself. Since the same word can appear multiple times within documents or even sentences, a **WordInstanceNode** is used to represent the unique occurrence of a word in a particular context. The identifier of a WordInstanceNode is used to tell it apart from other instances of the word, and can be therefore tagged with feature data such as tense, number, and part-of-speech. Given a word instance, a **ReferenceLink** is used to associate it with the underlying word.

  ```
  (ReferenceLink (stv 1.0 1.0)
      (WordInstanceNode "wordinstance@UUID")
      (WordNode "word")
  )
  ```

- For parses:

  Since different parses may assign different feature data to different word instances, each WordInstanceNode is associated with a particular parse by means of a **WordInstanceLink**.

  ```
  (WordInstanceLink (stv 1.0 1.0)
      (WordInstanceNode "wordinstance@UUID")
      (ParseNode "sentence@UUID")
  )
  ```

  As with SentenceLinks, this type of link only indicates if a given word instance is a part of the parse, but it doesn't indicate the structure within the parse. To represent the word order within a parse we use a **WordSequenceLink** with a NumberNode. Note again that '7' is only an example here too.

  ```
  (WordSequenceLink (stv 1.0 1.0)
      (WordInstanceNode "word@UUID")
      (NumberNode "7")
  )
  ```

- For word linkages:

  As mentioned in section 1.1, Link Grammar (LG) is a sentence-parser that creates undirected, planar link-graphs by connecting pairs of ordered words that follow certain relationship rules. Such rules are defined in a language grammar, where word-pair-links are grouped into categories and each link is identified with relation to these categories. LG is integrated and greatly used in OpenCog's projects. Following the same structures used before, LG links are represented in the Atomspace using an EvaluationLink predicate:

```
(EvaluationLink (stv 1.0 1.0)
    (LinkGrammarRelationshipNode "Rel-typ")
    (ListLink
        (WordInstanceNode "left-word@UUID")
        (WordInstanceNode "right-word@UUID")
    )
)
```

  Here *Rel-typ* is the name of the linkage yielded by the LG relation. Also, it uses an ordered list because even though links are undirected, word order does matter in most of the cases. Notice that once again word-instances are used instead of words because LG might output more than one valid parse and different parses can have different LG linkages.

With the purpose of illustrating these sentence representations we show an example next. More examples can be found in the wiki page of the project [38][12]. A text document named *example.txt* containing only the following line:

I saw it here

would be processed and saved in the structures bellow:

```
(SentenceLink
    (SentenceNode "sentenceone@2d19c7e7-2e02-4d5e-9cbe-6772174f3f4d")
    (DocumentNode "exammpledoc@308a2446-046c-4f00-bf8d-7e4d2f256875"))

(SentenceSequenceLink (stv 1 1)
    (SentenceNode "sentenceone@2d19c7e7-2e02-4d5e-9cbe-6772174f3f4d")
    (NumberNode "1"))

---------------------------------------------------------------------
(ParseLink (stv 1.0 0.83)
    (ParseNode "sentenceone@2d19c7e7-2e02-4d5e-9cbe-6772174f3f4d_parse_0")
    (SentenceNode "sentenceone@2d19c7e7-2e02-4d5e-9cbe-6772174f3f4d"))
```

[12] https://wiki.opencog.org/w/Sentence_representation

```
(WordInstanceLink (stv 1.0 1.0)
    (WordInstanceNode "I@52c30b4d-5717-47cb-822d-b2caa44f94b9")
    (ParseNode "sentenceone@2d19c7e7-2e02-4d5e-9cbe-6772174f3f4d_parse_0"))
(WordSequenceLink (stv 1.0 1.0)
    (WordInstanceNode "I@52c30b4d-5717-47cb-822d-b2caa44f94b9")
    (NumberNode "1"))
(ReferenceLink (stv 1.0 1.0)
    (WordInstanceNode "I@52c30b4d-5717-47cb-822d-b2caa44f94b9")
    (WordNode "I"))

(WordInstanceLink (stv 1.0 1.0)
    (WordInstanceNode "saw@0f223d17-31a6-49fe-9d37-350d50c53926")
    (ParseNode "sentenceone@2d19c7e7-2e02-4d5e-9cbe-6772174f3f4d_parse_0"))
(WordSequenceLink (stv 1.0 1.0)
    (WordInstanceNode "saw@0f223d17-31a6-49fe-9d37-350d50c53926")
    (NumberNode "2"))
(ReferenceLink (stv 1.0 1.0)
    (WordInstanceNode "saw@0f223d17-31a6-49fe-9d37-350d50c53926")
    (WordNode "see"))

(WordInstanceLink (stv 1.0 1.0)
    (WordInstanceNode "it@f6c2a2a2-232b-4e33-9c93-72f211b475d3")
    (ParseNode "sentenceone@2d19c7e7-2e02-4d5e-9cbe-6772174f3f4d_parse_0"))
(WordSequenceLink (stv 1.0 1.0)
    (WordInstanceNode "it@f6c2a2a2-232b-4e33-9c93-72f211b475d3")
    (NumberNode "3"))
(ReferenceLink (stv 1.0 1.0)
    (WordInstanceNode "it@f6c2a2a2-232b-4e33-9c93-72f211b475d3")
    (WordNode "it"))

(WordInstanceLink (stv 1.0 1.0)
    (WordInstanceNode "here@bf71826c-487e-42df-a941-0ecd3c942a76")
    (ParseNode "sentenceone@2d19c7e7-2e02-4d5e-9cbe-6772174f3f4d_parse_0"))
(WordSequenceLink (stv 1.0 1.0)
    (WordInstanceNode "here@bf71826c-487e-42df-a941-0ecd3c942a76")
    (NumberNode "4"))
(ReferenceLink (stv 1.0 1.0)
    (WordInstanceNode "here@bf71826c-487e-42df-a941-0ecd3c942a76")
    (WordNode "here"))

----------------------------------------------------------------------
(ParseLink (stv 1.0 0.17)
    (ParseNode "sentenceone@2d19c7e7-2e02-4d5e-9cbe-6772174f3f4d_parse_1")
    (SentenceNode "sentenceone@2d19c7e7-2e02-4d5e-9cbe-6772174f3f4d"))
```

```
(WordInstanceLink (stv 1.0 1.0)
    (WordInstanceNode "I@3fb606f8-0198-1668-e288-1de81ee1064a")
    (ParseNode "sentenceone@2d19c7e7-2e02-4d5e-9cbe-6772174f3f4d_parse_0"))
(WordSequenceLink (stv 1.0 1.0)
    (WordInstanceNode "I@3fb606f8-0198-1668-e288-1de81ee1064a")
    (NumberNode "5"))
(ReferenceLink (stv 1.0 1.0)
    (WordInstanceNode "I@3fb606f8-0198-1668-e288-1de81ee1064a")
    (WordNode "I"))

(WordInstanceLink (stv 1.0 1.0)
    (WordInstanceNode "saw@2307ab5f-af07-9642-21f6-d04d45e355f2")
    (ParseNode "sentenceone@2d19c7e7-2e02-4d5e-9cbe-6772174f3f4d_parse_0"))
(WordSequenceLink (stv 1.0 1.0)
    (WordInstanceNode "saw@2307ab5f-af07-9642-21f6-d04d45e355f2")
    (NumberNode "6"))
(ReferenceLink (stv 1.0 1.0)
    (WordInstanceNode "saw@2307ab5f-af07-9642-21f6-d04d45e355f2")
    (WordNode "saw"))

(WordInstanceLink (stv 1.0 1.0)
    (WordInstanceNode "it@b7505847-1b7c-82d3-6f5b-ededbeb85fe8")
    (ParseNode "sentenceone@2d19c7e7-2e02-4d5e-9cbe-6772174f3f4d_parse_0"))
(WordSequenceLink (stv 1.0 1.0)
    (WordInstanceNode "it@b7505847-1b7c-82d3-6f5b-ededbeb85fe8")
    (NumberNode "7"))
(ReferenceLink (stv 1.0 1.0)
    (WordInstanceNode "it@b7505847-1b7c-82d3-6f5b-ededbeb85fe8")
    (WordNode "it"))

(WordInstanceLink (stv 1.0 1.0)
    (WordInstanceNode "here@a8f70463-9441-d485-9711-a3d40a85e08f")
    (ParseNode "sentenceone@2d19c7e7-2e02-4d5e-9cbe-6772174f3f4d_parse_0"))
(WordSequenceLink (stv 1.0 1.0)
    (WordInstanceNode "here@a8f70463-9441-d485-9711-a3d40a85e08f")
    (NumberNode "8"))
(ReferenceLink (stv 1.0 1.0)
    (WordInstanceNode "here@a8f70463-9441-d485-9711-a3d40a85e08f")
    (WordNode "here"))
```

3
Methods

The ULL pipeline is divided into sub-processes that handle different chores or sub-modules, as explained in Chapter 1; from them, our thesis addresses corpora collection, parsing and word sense disambiguation. The task that the whole pipeline wants to tackle is not a simple one: the team building it is not certain if unsupervised grammar learning from unannotated corpora with neither grounding nor reinforcement feedback can succeed at all. To make the problem solvable at least to some extent, some simplifications are performed, and some methodologies are preferred over others. A simplification concerning the module addressed in this thesis (i.e. parsing) is that, instead of immediately processing text that resemble everyday language, we collected a series of controlled corpora of increasing complexity. This chapter explains the assumptions made for corpora. It explains as well the methodologies implemented for the parsing and pre-processing learning modules of the ULL pipeline. Finally, it explains the methods that will be used to evaluate the performance of our implementations applied to the aforementioned corpora.

3.1 Controlled corpora

Once completed, the ULL pipeline is expected to derive a grammar after processing large unannotated corpora. 'Unannotated' does not necessarily imply that the texts may not be controlled, at least during the pipeline construction and testing phases. The control that we implement on the learning texts within this thesis project relates to different aspects. One of the aspects refers to the content of the text; for example, we may clean corpus data to contain texts in one language only, as the difficulties of learning more than that would increase substantially. Another aspect relates to the complexity of the texts being inputted. For instance, we may limit the vocabulary size of the text, or consider sentence length and composition as other parameters to vary. This allows us to understand the performance of the algorithms in simpler scenarios, isolate potential issues with them from the intricacies inherent to more realistic language, and more easily find ways to overcome them.

If one looks back, the first task that we identified in section 1.2 as part of our contributions to the OpenCog ULL project was to find or create adequate text corpora to test our implementations. In this thesis we want our algorithm to learn the structure of a language in a manner resembling that of human children. Recall when a baby starts speaking: the first words are always objects, actions and simple concepts, like 'mom', 'eat' and 'yes'. Afterwards, they learn the relation between these frequently

used vocabulary, and only later add more complicated sentence structure, like the use of articles and adjectives or word order. Likewise, we elaborated and selected corpora that start with simple structures and limited vocabulary and then gradually increase their complexity.

Next we describe in more detail the corpora we will use, in the given order, for testing and evaluating our implementations, as described in sections 3.2, 3.3 and 3.4:

- Proof-of-Concept Turtle (**POC-Turtle**) corpus - a manually created corpus representing a closed semantic space of 16 different words communicated in the simple Turtle language [19]. It contains a total of 12 sentences with 4 words each, totalling 48 words[1]. This language is used in semantic web programming and is limited to three words per sentence in a strict Subject-Verb-Predicate triplet grammar. The complexity of such language can be thought closer to complexity of language that non-human primates can learn [20] or that children at age up to 3 can use [21].

  Example sentences:
  tuna isa fish .
  parrot has wing .

- Proof-of-Concept English with no ambiguity (**POC-English-NoAmb**) corpus - a manually created, closed semantic space of 19 English words with nearly the same frequency of use, communicated in simple grammatical constructions of 4-5 words per sentence. It contains a total of 36 sentences and 206 words, none of which is considered ambiguous for the purposes of this thesis.

  Example sentences:
  A daughter is a child .
  Cake is a food now .

- Proof-of-Concept English with ambiguity (**POC-English-Amb**) corpus - a manually created, closed semantic space of very few English words (43) with nearly the same frequency of use, communicated in simple grammatical constructions of 4-5 words per sentence. It is similar to the previous corpus (it is actually a superset of it) but contains two words involved in semantically and grammatically ambiguous constructions. Syntactic ambiguity is represented by the word 'saw', which can be either a noun or a verb. The word 'board' is used to exemplify semantic ambiguity, with two different noun meanings (surface to write on vs. group of people directing some activity). It contains a total of 88 sentences and 573 words.

[1] All corpus and vocabulary sizes are given after the corpus has been pre-processed for our purposes (see section 3.1.1). Words, punctuation marks and any other symbols appearing as separated tokens are considered words.

Example sentences:
Mom saw dad with a saw .
Dad is on the board of directors .

- **Child Directed Speech** (CDS) corpus - a sub-collection of the CHILDES corpus [32] containing adults' speech directed to children, characterized by adapted lexicon and reduced grammar complexity. Specifically, it contains the Brent-Ratner-corpus [6, 9] and the UCI-Brent-Syl-corpus [10][2]. It contains a total of 4k unique words in 38k sentences, for a total of 130k words.

  Example sentences:
  Did you wanna sit down ?
  And they all rolled over and one fell out

- **Gutenberg Children** corpus - a compendium of books for children contained within Project Gutenberg[3], following the selection used for the Children's Book Test of the Babi CBT corpus [25][4]. It contains a vocabulary of 40k words in 207k sentences, with a total size of 2.7M words.

  Example sentences:
  Alice led the way , and the whole party swam to the shore .
  " They're not in the past tense , " retorted aunt Jamesina .

- **Brown** corpus [20] - a 1-million word compilation of printed English prose from 1961, covering 15 different categories including Press, Fiction, Humor, among others[5]. After pre-processing it (see section 3.1.1), the corpus contains a total of 482k words distributed in 35k sentences, with a vocabulary size of 43k words. This corpus is considered 'natural' English, and is the most advanced one among our tested corpora.

  Example sentences:
  Only within the framework of a mature relationship characterized by honest appraisals of performance can we provide telling assistance .
  The battle of the Naktong River is just one example of how the battle cry and the spirit of The Fighting Seventh have paid off .

- NLTK's **Penn Treebank** Sample - a sample from the full Penn Treebank corpus [34], which is available through the NLTK suite [7]. It contains parse trees of 10% of the whole Wall Street Journal articles inside the original corpus. The corpus consists of a vocabulary of 6k words and a total of 2,285 sentences, for a total size of 38k words. Together with the Brown corpus, it represents the most advanced English used in this thesis. The purpose of

[2] Available at `https://childes.talkbank.org/derived/`
[3] `https://www.gutenberg.org`
[4] `https://research.fb.com/downloads/babi/`
[5] `http://www.sls.hawaii.edu/bley-vroman/brown_nolines.txt`

including a treebank is to use it as a Gold Standard during the evaluation of automatically-generated parses (see section 3.4).

Example sentences:
The problem involves the motion of small magnetic fields within superconductor crystals , limiting their current-carrying capacity .
The Underwood family said that holders of more than a majority of the stock of the company have approved the transaction by written consent .

3.1.1 Pre-cleaner

So far we described the content nature of the corpora. Nevertheless, another important aspect to take into account with regards to corpora control is their format. Collections of text like the ones just described normally come with a variety of symbols and content that are not exactly part of the language structure that we're interested in learning, e.g. HTML markup and graphics, variations of the same punctuation marks, UTF codes, capitalized letters, etc. In an ideal scenario, our language-learner should be able to also understand the function of that content, but at the current stage of the system it would benefit from using more standardized input. In order to ensure that the input is self-consistent and useful to our purposes, we pre-processed all the input corpora with a text-cleaning algorithm[6]. The pre-cleaner contains several options for pre-processing the input (which can be turned on and off) and can be grouped into the following broad categories:

- Character content of the text: lower-casing the corpus, standardizing different versions of some symbols (e.g. quote symbols), removing non-allowed characters, removing html markup, tokenizing special characters.

- Structure of the text: sentence splitter, header remover, maximum sentence and word lengths allowed.

- Encoding of entities: replacing numbers, URLs, dates, times with encoding tokens (e.g. '@url@'), decoding escaped html and UniCode symbols.

For the purposes of this thesis, we pre-process the corpora before parsing using the pre-cleaner default values, which imply the following actions:

- Determine sentence boundaries and split sentences into one per line.
- Remove headers (only for the Gutenberg Children corpus).
- Subtitute URL and e-mail addresses to special tokens. Same for dates, times and numbers.
- Decode escaped HTML and unicode symbols to their printable version.
- Standardize single quotes to one symbol. Same for double quotes, dashes, long dashes.

[6]Pre-cleaner available at: `https://github.com/singnet/language-learning/tree/master/src/pre-cleaner`

- Tokenize quotes, double quotes and several other punctuation marks.
- Remove asterisks, underscores, long dashes.
- Remove sentences with more than 25 tokens.
- Remove words with more than 25 characters.
- Convert text to lower-case.

3.2 Parsing

The grounding for our parsing efforts in the ULL project[7] is the unsupervised, mutual information-based parser originally proposed by Deniz Yuret [55], which was briefly explained in the previous chapter (section 2.1). The parsing pipeline consists of three stages: gathering of frequency information of words and word pairs from the training data, calculating the fractional mutual information (FMI) of those word pairs, and parsing of sentences using the estimated FMI and a Maximum Spanning Tree algorithm (MST). The counting and the parsing stages are independent and consequently can be performed using the same or different sets of training sentences. Each of the stages will be explained subsequently.

3.2.1 Frequency counting

Mutual information can be estimated by observing the frequency in which words and word pairs appear in the studied corpora (see section 3.2.2). In order to determine these frequencies, the system includes a text-processing pipeline that scans a text document assuming that each new line represents a new sentence; then it tokenizes the text, which in this case means breaking it into words; and finally it goes through the text again word by word, sentence by sentence, saving the encountered word instances and sentences in related atoms as described in section 2.3[8]. Counting of individual words in this manner is simple: it is possible to attach to each WordNode a count truth value (a numeric valuation for an atom as described in 2.3) that should be updated each time the text-processor runs into a corresponding WordInstanceNode. Nevertheless, in this project we are not interested on the appearance of words by themselves but on the presence of words in relation to other words. Thus we need a different kind of structure.

Recall that the Atomspace has a special structure to represent link grammar links. This structure uses a node with the name of the link type to connect two WordInstanceNodes corresponding to the appearance of a word-pair. Since we don't have any link types at this point and we are enticed to registering the repetitive presence of word-pairs (and not their individual appearance), one possibility is to modify this structure to use a generic link type and use WordNodes instead of WordInstanceNodes. The structure then would be as follows:

[7] Code hosted at `https://github.com/singnet/opencog/tree/master/opencog/nlp/learn`

[8] Unless we use LG, only one parse is assigned to each sentence and it is a sequential parse, since we don't have any parsing rules available at that point.

```
(EvaluationLink
    (LinkGrammarRelationshipNode "ANY")
    (ListLink
        (WordNode "left-word")
        (WordNode "right-word")
    )
)
```

Another option is to create a new link atom where the link type is simply named after its word-pair nature as follows:

```
(EvaluationLink
    (PredicateNode "*-Sentence Word Pair-*")
    (ListLink
        (WordNode "left-word")
        (WordNode "right-word")
    )
)
```

In either case, the presence and frequency ($N(x, y)$ and $P(x, y)$; see section 3.2.2) of these word-pairs in the text can be attached to these links as truth values. However, to detect these 'presences' we need to process the text in an observation mode that has a way to determine when the respective counts should be updated.

In this thesis project we considered three variations or modes for counting word-pairs:

- **LG-any**: In this mode, for each sentence we sample a limited number of parses using the Link Grammar Parser in random parsing mode ('any' mode), and then count the pairs of words linked in these parses. Hence, the structure used for keeping these counts is the first of the two mentioned above. The number of different parses per sentence to consider should be given as a parameter. Here we fixed it to 100.

- **Clique-WIN**: In this mode we use a Cartesian combination of per-sentence words that are within a given window size. Viz., per sentence, the observer iterates over each word and pairs it with every other word located within a determined maximum distance to its right. Distance in this context is defined as the difference between the words' positions in the sentence; as such, neighboring words have distance of 1. The window size or maximum distance between words in a pair must be given as a parameter to the observer. For our experiments we fixed it to 6, based in linguistic evidence that (most) syntactic information in a sequence of words lies within this distance [28]. The counts for this kind of pairs are held as values in the second type of word-pair structures mentioned above.

- **Clique-WIN-dist**: In this mode we use a Cartesian combination of per-sentence words that are within a given window size, and account for pair distance. That is, we do the same as in the Clique-WIN method, except that now each word-pair is counted a number of times determined by the: distance (as defined in Clique-WIN) between the words. In the Clique-WIN method in every iteration that we pair up words we increase their count by one; in this method we increase their count by the quotient WIN/d, where WIN is the window size and d is the actual distance between the words, thus favoring words that appear closer together in a sentence. Once again, the window is set to 6 and the structure used to keep the counts is the second of the ones described above.

3.2.2 Mutual information calculation

Lexical attraction was defined in section 2.1 as the MI held in a syntactic relation. However, because the appearance of a specific word in particular place in a sentence is not the random variable itself but an instance of it, what Yuret [55] really calculated in his parser was the FMI of the word pairs. Furthermore, we are interested in ordered word pairs, not individual words, so in the context of this thesis the random variables used for defining the FMI of an ordered word pair are as follows:

- X is the random variable of the event of having a word x related to any other word that is at its right in a sentence. Let * be a wild-card, i.e. any word in the vocabulary of a language, then X represents the event of observing the ordered word pair $(x, *)$.

- Y is the random variable of having a word y related to any other word that is at its left in a sentence. Similarly, Y represents the event of observing the ordered word pair $(*, y)$.

Moreover, if a word x has another word y to its right, then the word y has word x to its left; and $p(X = x, Y = y) = p(Y = y, X = x)$ is the probability of observing the word pair (x, y) in exactly that order. The random variables X and Y do not have disjoint sample spaces given that words can be syntactically related to both, words at their left and right in a sentence. Hence, to avoid confusion, we will set the notation $P(x, y)$ to be the probability of observing the word-pair (x, y) in a language in that same order[9]. Using this notation and the notion of a wild-card explained before, we can rewrite the equation to calculate the FMI of a word pair (x, y) as:

$$FMI(x, y) = \log_2 \frac{P(x, y)}{P(x, *)P(*, y)}$$

This obviously implies that $p(X = x) = P(x, *)$ and $p(Y = y) = P(*, y)$. Also notice that $P(x, y)$ and $P(y, x)$ represent two different things.

[9] In this sense, $P(y, x)$ would be without ambiguity the short for $p(X = y, Y = x)$.

Considering that it is impossible to know the relative occurrence of ordered word-pairs in an entire language, we will use the word and word-pair counts collected from observing texts (as described in the previous section) to estimate it. After running the text-processor over a corpus, the Atomspace will gather statistics and hang them in the created sentence structures. In particular, it will contain the number of times each ordered pair (x, y) was observed in the text, which we will note as $N(x, y)$[10]. Using these, we can estimate the value of $P(x, y)$ as:

$$P(x, y) \approx \frac{N(x, y)}{N(*, *)}$$

where $N(*, *)$ is the total number of ordered word-pair observations. In turn, this last one can be computed as:

$$N(*, *) = \sum_{x,y} N(x, y)$$

At last, we just need to calculate the marginal probabilities. These too can be estimated using the word-pair statistics gathered:

$$P(x, *) = \sum_{y} P(x, y) \approx \frac{N(x, *)}{N(*, *)}$$

$$P(*, y) = \sum_{x} P(x, y) \approx \frac{N(*, y)}{N(*, *)}$$

Here $N(x, *)$ is the total number of times the word x was observed paired to the left of any other word in the text; and it can be computed as:

$$N(x, *) = \sum_{y} N(x, y)$$

Similarly, $N(*, y)$ is the number of times the word y was observed paired to the right of any other word in the text; and it can be computed as:

$$N(*, y) = \sum_{x} N(x, y)$$

Replacing the above computations of wild-card probabilities we can estimate the fractional mutual information of an ordered word pair (x, y) as:

$$FMI(x, y) \approx \log_2 \frac{N(x, y)N(*, *)}{N(x, *)N(*, y)}$$

Finally, recall that FMI is a ratio that compares the probability of observing the two words together in a specific order vs observing them separately, and therefore lies in the range $(-\infty, \infty)$.

[10] Note that $N(x, y)$ is absolutely not the same as $N(y, x)$ either.

3.2.3 MST algorithm

Once we have a set of ordered word-pairs together with an estimation of their pointwise mutual information, we can proceed to parse the text making use of these statistics. As we described in the introduction, the Opencog project uses an algorithm for maximal parsing, but it is not any of the two illustrated by Yuret's work and mentioned in section 2.1. The reason is that the optimal algorithm has a high complexity to be useful, and we are still willing to sacrifice a little performance efficiency (i.e. a little worse than $O(n^2)$) in order to get more trustful results. Thus, an MST (minimum spanning tree) algorithm was developed within the team, which maximizes the total score of the tree[11] instead of minimizing it, and restricts the result to a projective tree as well[12].

When it comes to projective maximal parsing algorithms, the state-of-the-art best rated one is Eisner's, which runs in $O(n^3)$ time [35]. Nonetheless, it uses dynamic programming and it's not the most straightforward to implement within OpenCog. The simplest implementations of MST algorithms are the three classical ones: Prim, Kruskal, and Borůvka's [3, 43]. On one hand, modifying Prim to restrict crossing edges would be simple, but this modification will make the outcome of the algorithm heavily dependent on the initialization, which didn't seem very promising[13]. Kruskal on the other hand, conveys the impression of being too difficult to enforce with a no-link-cross constraint. Hereinafter, the algorithm we are using is an implementation of a variant of Borůvka's algorithm, which seems like the most robust and fast enough alternative for the project's needs.

Overall, this algorithm takes a sequence of atoms representing an ordered list of words and finds an unlabelled, undirected, projective dependency parse for it. This means that we need a parser-method that feeds the algorithm with such a sequence. Therefrom, we need to process the input text once more, tokenize each sentence into a list of words, and create a sequence of atoms per sentence from the word list (the atoms used here are of the WordNode type). The parser-method then takes each sequence and creates a set of pairs (*number.atom*) where *number* corresponds to the position of the respective word in the sentence. Finally it inputs this set as the set V of vertices of a clique graph $G = (V, E)$[14], together with the respective set of edges E and a predefined edge scoring function $f : E \to \mathbb{R}$, to the modified MST algorithm which finds a projective tree T in G with maximal total score $F = \sum_{e \in T} f(e)$. A pseudo-code of such modified MST algorithm is described below in Algorithm 1. The algorithm returns the tree as a list of atom-pairs, each with its associated score.

[11] The total score is the sum of the weights of each edge of the tree, as defined in any classical MST algorithm [43].

[12] A projective tree is one such that if we number the vertices and arrange them in order in a line, the edges of the tree can be drawn above the vertices without crossings [35].

[13] The success of the original Prim algorithm lies in the fact that if there is an optimal solution, the algorithm will find it regardless of which is the first vertex chosen. That fact no longer holds if we add the projectiveness restriction.

[14] A clique is a graph where there exists an edge between every possible pair of vertices [43].

Algorithm 1: Pseudo-algorithm for modified MST parser

```
Input  : A clique graph G=(V,E), where V is a linearly ordered set, and an
         edge scoring function f.
Output: The set L of edges representing the links in the parse.
Take the edge c=(a,b) from E that has the highest score f(c);
Initialize a set L of links assigned to the parse to contain only edge c;
Initialize a set U of disconnected vertices to contain V-{a,b}, that is, all the
 vertices of the graph except the ones adjacent to edge c;
Initialize the set C of connected vertices to be {a,b}, that is, a set with just
 the vertices incident to edge c;
while set U is not empty do
    Let O be the set of all edges with one endpoint in U and the other in C;
    Initialize the candidate edge v to be nothing;
    while O is not empty AND v is nothing do
        Take the edge x from O that has the highest score f(x);
        if x does not cross any links in L then
            Let v be x;
        else
            Remove v from O;
        end
    end
    if v is nothing then
        Exit the loop;
    else
        Let v=(u,w) where u is in U and w in C;
        Add v to L;
        Add u to C;
        Remove u from U;
    end
end
```

The scoring function we input to the algorithm is very important, because this numeric score is the one being maximized during the parse. It should be a function that receives as input an edge e, such that the distance $d(e)$ between its two endpoints is known, and outputs a real number. In the case of our structure, the distance is simple to calculate given that our vertices are pairs (*number.atom*). Let $v = ((x, a), (y, b))$ be an edge that connects word-atom a that is in position x in the sentence with a different word-atom b that is in position $y \neq x$ in the sentence, then such distance is $d(v) = |x - y| > 0$. A very simple scoring function could just return that distance, but we are interested in a function that favors links with meaningful relations. Thus, in this thesis we experimented with two scoring functions that use both the distance and FMI of word pairs:

- **FMI**: since we are interested in links with high lexical attraction, we use the point-wise mutual information between the two words the edge is trying to

connect. We also don't want links that are too long so we will give a scoring penalty to links that have $d > 16$. Thus, this scoring function will return the FMI to every word pair where it is defined, and a very negative value when it is not or when the distance between the words is greater than 16.

- **FMI-dist**: this scoring function does the same as the one above, except that we account for distance a little further. We don't only want links between words that are no further apart than 16 places from each other, but we also want to favor links among words that are closer together in general. Thus we will add to the point-wise mutual information of the pair an additional value that is inversely proportional to the distance between the pair. Specifically, when defined and within a distance of 16, the scoring function will output $FMI + 1/d$, or a very negative value otherwise.

Finally, although the concept is very easy to understand, it is important to define what it means for an edge to cross any links. Recall that V is linearly ordered, so a relation $<$ among the vertices should be defined. Let the edge be $e = (j, k)$ such that $j < k$, for any $l \in L$ with endpoints x and y such that $x < y$, we say e crosses l if one of the two is true:

- $j < x$ AND $x < k$ AND $k < y$
- $x < j$ AND $j < y$ AND $y < k$

3.3 Word-sense Disambiguation

After the input sentences are organized in parse structures, the ULL pipeline continues by trying to automatically find categories for these words, based on the obtained parses. Hence, the quality of the parses that our algorithm outputs will be very influential in the output of the entire syntactic loop. Nonetheless, as mentioned in section 1.3, we believe that the outcome of a single pass through the loop is expected to be of low quality because it is just a first learning step. The idea is that at the end of every cycle, the process should receive the output information from the previous cycle or from the other language cycles, and use it to help improve the quality of the output of the next cycle. As a matter of fact, our hypothesis is that feedback is essential for unsupervised learning. Therefore, in each recursive pass, we expect that if we change the definition of our new 'objects' to be more refined[15], thus raising the quality of the input, the quality of the output should improve as well.

Even though we can't evaluate the entire loop, we still wanted to experiment with varying inputs to see if these will result in varying outputs up until the parser. Recall from section 3.1 that some of our corpora contain different uses of ambiguous words. On the first experiments of our thesis, these words are fed into the pipeline without any distinction, which implies that all the statistics gathered within the parser

[15] For instance, add features to the input words, or take word categories or grammar rules to be the new objects instead.

are done considering the word to be the same in all contexts. For example, all the frequency counts for the different word senses get attached to the same WordNode, causing the marginal counts for the different senses to be higher than they actually are, and the usage statistics for them to be aggregated. If we could detect in a text which words are ambiguous and identify them before passing them to the parser, the statistics gathered during the process would change, possibly varying the parser outputs too. This is what we want to explore in this section: disambiguation of word-senses as a way of redefining and improving the quality of our input 'objects'.

Moreover, in section 2.2, we described how Adagram builds a space of learned distributed representations obtained from the context of each word. AdaGram also offers the additional benefit of disambiguating polysemous words while learning those representations. The ULL pipeline could naturally benefit from this capability by using the vector space built with AdaGram, since it would contain different embeddings for ambiguous words, which could be directly categorized in different groupings by the clustering algorithms down the pipeline. However, here we will just use AdaGram as a way of 'refining' our input by pre-processing the ambiguous corpora before MST-parsing. In order to do so we need a way of evaluating how much the disambiguation improve the quality of our input. One possibility is to simply apply this algorithm to our different training corpora and compare the results obtained by using different parameters to a known standard of word similarity.

The way we proceed to do this is the following. First we train AdaGram models with our corpora using different hyper-parameters, and find the one that performs best in a Word Sense Disambiguation task. To evaluate the quality of our model intrinsically we use a Word Similarity Evaluation metric and compare results to the literature (see section 3.4.2). Then, we use the chosen model to annotate each ambiguous word in the original corpus with the sense most appropriate to its context. As an example, for the polysemous word 'saw', we might end up with different senses: 'saw@a' 'saw@b'. Because every process in the pipeline is language-independent, we could continue by feeding the MST-parser and the subsequent ULL modules with a corpus where words are replaced with their unambiguous versions, hoping to end up with a more accurate grammar. Even though the scope of our thesis reaches the parsing module only, we can still evaluate extrinsically the selected model by comparing directly the outputs of the parser using the disambiguated text versus using the original one.

Moreover, AdaGram lets you select the values of a range of hyper-parameters for training an embeddings model. Among them, we selected the ones that we considered relevant for our experiments, guided by the results and discussion by Bartunov and colleagues [2]. The following list describes the hyper-parameters explored in AdaGram, as well as in the code we wrote to annotate the original corpus[16]:

- Dimensions: the length of the embedding vectors in the model.

[16] Code available at `https://github.com/singnet/language-learning/blob/master/src/word_senser/annotate_corpus.jl`

- Epochs: the number of training epochs for the model.
- Min-freq: the minimum number of times a word needs to appear in the corpus to be considered for training.
- Remove-top: defines how many of the most-frequent words (in order) are ignored from the corpus for training purposes.
- Alpha: regulates the number of senses that each word will have, together with the frequency of that word in the corpus.
- Window-model: The maximum size of the context window for each word instance during training (see Figure 2.2).
- Window-annotate: The size of the context window for each word instance during the annotation step.
- Min-prob: The minimum prior probability for at least two senses of a word for it to be considered ambiguous during annotation.

Given the large hyper-parameter space available for AdaGram and our annotator, it is not trivial to figure out which of them cause a larger impact on the quality of the trained models. To simplify this task we used Altair's HyperStudy[17], a specialized software for Design Exploration and Optimization which, among other managing and statistical tools, is able to calculate the effect that a set of variables have in the experiment's output(s), run optimization algorithms on the relevant variables and find an optimal model. To guide the hyperparameter search, we use the disambiguation metrics detailed in the upcoming evaluation section 3.4.2; once found, we use the selected model to annotate the original corpus for subsequent extrinsic evaluation.

For corpus annotation using an AdaGram model, we take each word in the corpus and check if it is represented in AdaGram as ambiguous, i. e. if it is represented by more than one vector whose prior is above the hyperparameter 'Min-prob'. If that is the case, we take the word's context (defined by hyperparameter 'Window-annotate') and use AdaGram's disambiguation capabilities[18] to determine what is the most likely prototype that represents that use of the word, and annotate it with a tag representing that prototype (we use the symbol '@' followed by single letters as tags). If successful, the result of this process is a corpus in which ambiguous words have been disentangled into their different senses.

Finally, there is one more aspect to consider. The way the original AdaGram algorithm works, as described in section 2.2, is by taking a continuous block of text

[17] `https://altairhyperworks.com/product/HyperStudy`

[18] Using the `disambiguate` function, see AdaGram's API description: `https://github.com/sbos/AdaGram.jl/wiki/API`

and building training samples by assigning to each word a number of neighboring words as context. The number of neighboring words is determined by a fixed window size, which is a hyperparameter of the system. This implies that for words located near sentence boundaries (closer than the window size, to be precise), some of its neighboring words will not be part of the same sentence, but rather of a previous or following one. However, in this thesis we are interested in using AdaGram to disambiguate words based only on the sense they have within a sentence. Consequently, we modified AdaGram to take individual sentences as input (instead of a continuous block of text) and to build training samples for a word by using only those neighboring words that are inside the same sentence (and the specified window size, of course)[19]. This will certainly reduce the number of training samples to some extent, but we expect that these will be of better quality and provide better results.

3.4 Evaluation

3.4.1 Parsing

Using the MST pipeline described in 3.2, we generate dependency trees for the sentences in each of the aforementioned corpora (section 3.1). We do it by using different pair-counting methods and scoring functions. A total of 6 different combinations of these counting-methods and scoring-functions are possible. Because of the varying nature of the corpora, a different subset of those combinations is used to parse each corpus. Section 4.1 shows which combinations are used for each corpus. Many of these combinations usually result in different parsing outputs and we need a way of comparing these outputs.

In order to evaluate the quality of these unsupervised parses, we need some standards and baselines for comparison. The ones we use in this thesis are described next:

- **Gold standard**: These are human-created or human-supervised parses, available for selected corpora. We manually crafted parses for the POC corpora, and obtained the free section of the Penn treebank, as mentioned in section 3.1.

- **Silver standard**: These are parses generated automatically by the LG parser in English. This last uses human-crafted rules to determine valid linkages.

- **Random parsing baseline**: This is a random planar tree structure connecting all words in the sentence. We generate this parse tree using the LG-parser in mode 'any', that is a parser without any grammar rules. An example is shown of such a parse in figure 3.1 where the arcs above the words represent the links among them.

- **Sequential parsing baseline**: This is a trivial 'each-word-links-to-the-next' parse tree. An example of such a parse is shown in figure 3.2. We use the

[19] Code available at: `https://github.com/glicerico/AdaGram/tree/take_sentences`

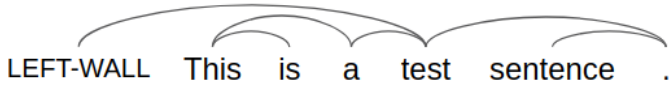


Figure 3.1: An example of a random parsing.

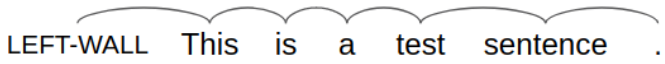


Figure 3.2: An example of a sequential parsing.

same notation as LG to allow for visual comparison.

The parses obtained with the MST-parser are then compared against at least one of the two standards (gold or silver). The idea to include a silver standard comes as a solution to the lack of gold standards for all of our corpora: it's important to compare the quality of our generated parses against some reference, even for corpora whose parses have not been curated by humans. The evaluation proceeds sentence by sentence, by taking all the links included in its parse as given by the standard, and calculating precision and recall for the MST-parse. Precision (P) and recall (R) are defined as:

$$P = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{TP + FN},$$

where TP stands for true positives, FP for false positives, and FN for false negatives. True positives are the links that exist both in the standard and the MST-parse. False positives refer to the links found by our method but not present in the standard. False negatives are the links listed in the standard parse but not found by the MST-parser. A graphic depiction of precision and recall is given in figure 3.3.

Once P and R are calculated, we can obtain a scoring metric from them. The metric we use is the F_1 which is commonly used in identification tasks. It is defined as:

$$F_1 = \frac{2 * P * R}{P + R},$$

This score is calculated for each of the parses of one sentence. We report the average of the F_1 score of all parses in a corpus. Also, we generate the baselines listed earlier (random and sequential) for each sentence in the corpora and evaluate them against the same standard(s). We do this in order to get an idea of the performance of our method against such uninformed ways of parsing. Since the random baseline involves fluctuations in F_1 score, we report the average evaluation of 10 independent

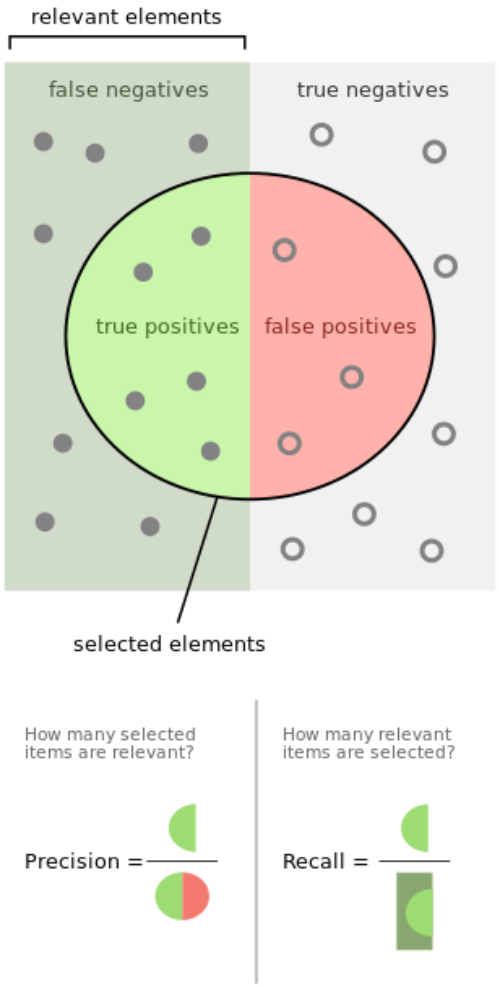


Figure 3.3: F_1 score metric: graphic interpretation of precision and recall. Image credit: Walber [51].

.

generations of this baseline.

Finally, one important thing to consider is that Link Grammar parser in English mode, the generator of our Silver standards, includes knowledge about the root of the tree (called the LEFT-WALL in LG) and about the final dot in a sentence. Our MST-parser is not designed to obtain any knowledge of these items. For this reason, we include an option in our evaluator to filter and ignore all links which include either the tree's root or the final dot of a sentence (if there is one). We activate this option when reporting our results.

3.4.2 Word-sense disambiguation

We also ran experiments where we use AdaGram to disambiguate word senses before parsing, as a pre-processing step, with the aim of improving the quality of the input to the parsing pipeline and consequently the parse quality. In order to use Adagram, we need to tune the hyper-parameters of the network for a given corpus to boost the model's performance. To do so, we build a disambiguation test set that consists of a number of sentences containing words considered ambiguous, and also manually annotate them to create a disambiguation Gold Standard. The idea is to compare the results of disambiguating a text using AdagGram against the GS to determine the model's disambiguation capabilities. We then use this capability assessment as our objective function when trying to vary the hyper-parameters of the model.

We evaluate the model's disambiguation performance with a metric commonly used for classification problems called 'pair-wise f-score' ($PWFS$) [33]. For each polysemous word in our test corpus, we calculate $PWFS$ and average among them. This metric uses the familiar concepts of precision and recall, but compares pairs that belong to the same class, instead of individual examples as seen in 3.4.1. Likewise, the $PWFS$ is defined as:

$$PWFS = \frac{2 * P * R}{P + R},$$

where P and R are precision and recall. This in turn are now defined as:

$$P = \frac{|F(K) \cap F(S)|}{|F(K)|}$$

$$R = \frac{|F(K) \cap F(S)|}{|F(S)|}.$$

where $F(K)$ is the set of all pairs that share class in the test data, and $F(S)$ represents the same for the gold standard.

As a secondary metric, we use 'overdisambiguated' words: words that have only one sense in the gold standard, but are considered polysemous by the embedding

model. This can help us decide between two different models that achieve similar $PWFS$, but where one indiscriminately assigns different senses to all words in the vocabulary. We define overdisambiguation as the fraction (OF):

$$OF = \frac{OW}{T},$$

where OW is the number of overdisambiguated words, and T is the size of the test corpus vocabulary.

After tuning the hyper-parameters for the corpus, we proceed to perform an intrinsic evaluation of the quality of the representations learned by AdaGram with a Word Similarity Evaluation. Given that AdaGram generates a real-valued vector representation for each discovered sense of a word, we can evaluate these representations with similarity calculations using vector algebra. For such task, we decided to use the Stanford Contextual Word Similarity (SCWS) dataset [27], which offers human-rated similarities of words within a context[20]. This database contains pairs of words, each of which is presented within its own context. Ten different native English speakers were independently asked to judge how similar the two words are in the given contexts using a scale from 0 (completely different) to 10 (completely the same). The database presents each pair of words (as well as their POS) with its individual similarity scores and their average.

In order to use SCWS to evaluate our models, for each word pair present in both the model and the dataset we calculate four similarity scores commonly used in the literature and summarized by Camacho-Collados and Pilehvar [12]. In the following description of the metrics, the set of senses for a word w_i is given by S_{w_i}, and $K_i = |S_{w_i}|$ is the number of senses for the word. The vectorial representation of a word sense s_j for word w_i is described as $\vec{s_j}$, where $s_j \in S_{w_i}$. We employ the cosine between two vectors v_1 and v_2 as a distance measure in all similarity scores: $\cos(\vec{v_1}, \vec{v_2})$. For each word pair (w_1, w_2), the similarity scores can be described as follows.

- **AvgSim** - Disregarding context, it calculates the distance between the vectorial representation of each of the senses of w_1 against all of those for the second word, and reports the average of all of them. Mathematically this can be calculated as:

$$AvgSim(w_1, w_2) = \frac{1}{K_1 K_2} \sum_{s_i \in S_{w_1}, s_j \in S_{w_2}} \cos(\vec{s_i}, \vec{s_j})$$

- **MaxSim** - Same as AvgSim, but it returns the maximum similarity between word vectors instead of the average. Mathematically this can be calculated as:

[20] Dataset available at `https://www.socher.org/index.php/Main/ImprovingWordRepresentationsViaGlobalContextAndMultipleWordPrototypes`

$$MaxSim(w_1, w_2) = \max_{s_i \in S_{w_1}, s_j \in S_{w_2}} \cos(\vec{s_i}, \vec{s_j})$$

- **AvgSimC** - As a contextual score, it needs a pre-disambiguation step performed by feeding a word (w_i) and a subset of its context (c_i) to the `disambiguate` function of AdaGram referred to in section 3.3, to obtain the likelihood that each available prototype ($s_j \in S_{w_i}$) is the right sense given the context: $P(s_j|w_i, c_i)$. Using this estimation, it performs a weighted average of the similarity between each pairing of the senses for one word with those of the other. Mathematically it is calculated as:

$$AvgSimC(w_1, w_2) = \frac{1}{K_1 K_2} \sum_{s_i \in S_{w_1}, s_j \in S_{w_2}} P(s_i|w_1, c_1) P(s_j|w_2, c_2) \cos(\vec{s_i}, \vec{s_j})$$

- **MaxSimC** - Using the same pre-disambiguation step as AvgSimC, it reports the distance between the most likely sense for each word, given their contexts. Mathematically it is calculated as:

$$MaxSimC(w_1, w_2) = \cos(\vec{s}_{max,1}, \vec{s}_{max,2}),$$

where

$$s_{max,i} = \arg\max_{s_j}(P(s_j|w_i, c_i)|s_j \in S_{w_i})$$

In our evaluation algorithm[21], if a word in a word-pair is not represented in a given model, that pair is skipped and ignored when counting. In a similar fashion, non-modelled words are ignored from the context of the target word during disambiguation. Each of these types of scores is then compared to the average human-similarity provided in the database by using Spearman's correlation coefficient (ρ) [53], which offers a measure of how well the ordering of the scores for the word pairs compare between the reference set (i.e. the SCWS dataset) and the test set (i.e. the model's similarities). If the two sets of results follow exactly the same rank order, then $\rho = 1$, while if they are inversely correlated, then $\rho = -1$. Also, if there is no correlation at all between the sets we have that $\rho = 0$.

Lastly, to be able to use this intrinsic evaluation method, we need our models to contain at least some of the words in the word-pairs used by the evaluation database. Hence, this method of evaluation does not hold for our POC corpora, which contain a minimal vocabulary. As a consequence, we will only use the POC-English with ambiguation corpus to take some illustrative examples.

[21] Available at: `https://github.com/singnet/language-learning/blob/master/src/word_senser/evaluate_SCWS.jl`

Additionally, we will evaluate the results of disambiguation extrinsically via comparing the parse quality obtained using the annotated corpora as input against using the original corpora with the same parsing methods. For this last evaluation we will use the same metrics explained in section 3.4.1.

4

Results

In this chapter we present the results obtained after implementing the algorithms described in chapter 3. As usual, we divide the chapter into one section for the parsing results and another one for the word-sense disambiguation. Analysis of the obtained results is provided as well.

4.1 Parsing

All of the corpora described in section 3.1 were used to test our implementation of the parsing pipeline. The results are summarized per corpus in the following subsections, each including example parses and tables showing the evaluation metrics. The notation used in the column headers of the evaluation tables is as follows:

- Parsing method: the technique used to parse the corpus as described in section 3.4.1. It can be one of the following: human parsing 'by hand' (GS), the Penn treebank (GS), LG-English (SS), random parsing, sequential parsing, or our implementation in OpenCog.

- Counting mode: the strategy used to count word pairs. It is one of the following options as described in section 3.2: using link grammar random parses (LG-Any), making a clique graph of the words within a distance window (Clique-WIN), or using a clique graph within a distance window and with distance weighing (Clique-WIN-dist). It applies to our implementations only.

- Scoring function: real valued function used to determine the weight of the links for the MST-parser. It is one of the two: fractional mutual information (FMI) or fractional mutual information with distance accounting (FMI-dist).

- F_1 Gold: evaluation result of parsing with the current settings against GS parsing. The metric used is F_1 Score.

- F_1 Silver: evaluation result of parsing with the current settings against SS parsing. The metric used is F_1 Score.

Here GS and SS are short for Gold Standard and Silver Standard respectively. By definition, these standards will achieve perfect F_1 against themselves. For reference, in the tables we will also compare GS against SS (when possible), and vice-versa.

4.1.1 POC-Turtle

As noted earlier, our POC-Turtle corpus contains language that is not proper English. This means that the LG-English parser would not be able to process it correctly. Thus, we created no SS for it, and we only compare our results against the GS. For this corpus, we used all parameter combinations and obtained the results presented in Table 4.1.

Table 4.1: Parsing results for POC-Turtle.

Parsing method	Counting mode	Scoring function	F_1 Gold[%]
Manual (GS)	–	–	100
Random	–	–	54
Sequential	–	–	100
OpenCog	Clique-WIN-dist	FMI-dist	**92**
OpenCog	Clique-WIN-dist	FMI	67
OpenCog	Clique-WIN	FMI-dist	50
OpenCog	Clique-WIN	FMI	50
OpenCog	LG-Any	FMI-dist	**92**
OpenCog	LG-Any	FMI	67

The first thing to notice from these results is that the random baseline, shown on a blue background for easy identification, barely scores above 50%; while the sequential baseline, shown with pink background, matches the GS exactly. This is a consequence of the fact that for this simple grammar the parses contain just links with one word connected to the next. When comparing the parses generated with our pipeline, we notice that the LG-any and Clique-WIN-dist counting modes outperform Clique-WIN. The best-scoring parses (presented in green font) are really close to the SS, and both of them use the FMI-dist scoring function. This suggests that accounting for distance in the parser is necessary when we want to get linearly related sentences.

Overall this corpus is too small (recall from section 3.1 that it contains only 12 sentences) for our evaluation results to be statistically significant. Nonetheless, this 'toy example' is very useful to illustrate how our parser works. For instance, if we take a close look at the outputs for the winner parsers (using the FMI-dist scoring function and either Clique-WIN-dist or LG-Any counting mode), there is something interesting to point out. There are only two sentences that are parsed incorrectly and both of them contain a word that is only used once in the text, in ditto sentence. This evidently affects the calculation of the FMI for pairs involving these words, resulting in parses that favor connections between sparse content words, which not even accounting for distance could break.

For example, one of the incorrect parse outputs that all of the OpenCog parsers share is for the sentence - *wing has feather .* -. The word *feather* appears only once in the text: in this sentence. The output of the parsers is shown in code below,

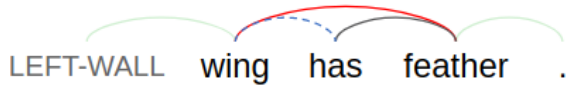


Figure 4.1: Example of an incorrect parse. Corpus: POC-Turtle. Method: OpenCog (all 6) against GS.

where the first line of the code is the sentence itself and each of the following lines contain one link of the parse (relation between two words) in the format:

$$number_1 \quad word_1 \quad number_2 \quad word_2$$

with the number of the position in the sentence preceding its word. A graphical interpretation of the same output parse is shown in Figure 4.1. In the figure, the continuous arcs represent the links that the parser outputs: black arcs represent correct links (with respect to the standard), red arcs represent incorrect links, and green arcs represent links that are ignored in the evaluation; and the discontinuous arcs represent missing links (i.e. links that appear in the standard but are not found by the parser). This same notation will be maintained for all of the remaining codes and figures in this section.

```
wing has feather .
0 ###LEFT-WALL### 1 wing
1 wing 3 feather
2 has 3 feather
3 feather 4 .
```

As one can notice in this example, the word *wing* is connected to the word *feather* instead of the word *has*. Observing the FMI calculated for these two word pairs help clarify this phenomenon. Table 4.2 shows such FMI for the three different counting methods. It is effortless to see in this table that the difference between the FMI of the two word pairs is always bigger than 1, independently of the counting-method chosen. Recall that frequent words provide less information content than infrequent words. Observing the FMI for other word pairs with *feather* we find that they are all similarly high, which is not the case for word pairs with alternative words that appear more frequently throughout the text (as we will see in the next examples). Since $1/d$ is always lower than 1, an implication of this is that accounting for distance in the scoring function of the parser is not enough to favor links among closer word-pairs (like the ones containing function words) against links that include infrequent content words.

Table 4.2: Example of pair FMI calculation in POC-Turtle: word *wing*.

Word Pair	Clique-WIN-dist	Clique-WIN	LG-Any
(wing, has)	0.5755	0.3219	0.4997
(wing, feather)	1.8710	2.3219	1.5939

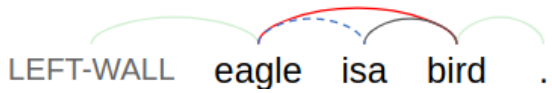


Figure 4.2: Example of an incorrect parse without distance accounting in scoring function. Corpus: POC-Turtle. Method: OpenCog (all 3) with FMI against GS.

Moreover, to understand why some of the methods under-perform others, it assists to portray few examples more. For the sentence - *eagle isa bird .* -, the winner parsers output the correct result, but all of the other methods output the parse shown in code next and illustrated in figure 4.2. Likewise, table 4.3 shows the FMI for the two word pairs involved in this parse and the three different counting methods. Comparably to the previous example, here the word *eagle* is connected to the word *bird* instead of the word *isa*. Obviously this means that the FMI for the word pair $(eagle, bird)$ is higher than the one for the word pair $(eagle, isa)$, independently of the counting method. However, this time the difference between the FMI of the two word pairs for the Clique-WIN-dist and the LG-Any methods is not bigger than 0.5, which allows distance accounting in the scoring function to break these incorrect links in favor of closer word links (because it adds $1/d = 1$ to $(eagle, isa)$, but only $1/d = 0.5$ to $(eagle, bird)$). This is due to the fact that the word *bird* appears more frequently in the text than the word *feather*. Given that there a many other examples akin to this one, it also explains why FMI-dist performs better for these two methods.

```
eagle isa bird.
0 ###LEFT-WALL### 1 eagle
1 eagle 3 bird
2 isa 3 bird
3 bird 4 .
```

Table 4.3: Example of pair FMI calculation in POC-Turtle: word *eagle*.

Word Pair	Clique-WIN-dist	Clique-WIN	LG-Any
(eagle, isa)	1.2035	0.7370	1.3557
(eagle, bird)	1.4990	1.7370	1.4499

Finally, let's analyze why the Clique-WIN counting method performs the worst of all. Below is the parse output for this method for the sentence - *herring has fin .* -, which is one out of a group of several common and alike examples. This parse is illustrated in figure 4.3. All of the other methods output the correct parse. Table 4.4 shows the FMI for the two word pairs involved in this parse and the three different counting methods. In this parse the word *herring* is connected to the word *fin* instead of the word *has*, which means that the FMI for the word pair $(herring, fin)$ is higher than the one for the word pair $(herring, has)$ for the Clique-WIN counting method. Unlike in the previous examples though, this is not the case for the other counting methods. In the case of Clique-WIN-dist the result is not surprising because the frequency for $(herring, has)$ pair will be counted with more weight.

Figure 4.3: Example of an incorrect parse without distance accounting in the counting method. Corpus: POC-Turtle. Method: OpenCog Clique-WIN (with both scoring functions) against GS.

However, this does suggest that LG-Any has an implicit distance accounting involved in its counting procedure that Clique-WIN does not have, which makes sense because longer links are more likely to cause link crossings. Also, notice that the difference in FMI for these two pairs with Clique-WIN is bigger than 1/2, which explains why there is no difference between the scoring functions for this counting method.

```
herring has fin.
0 ###LEFT-WALL### 1 herring
1 herring 3 fin
2 has 3 fin
3 fin 4 .
```

Table 4.4: Example of pair FMI calculation in POC-Turtle: word *herring*.

Word Pair	Clique-WIN-dist	Clique-WIN	LG-Any
(herring, has)	1.2035	0.7370	1.3557
(herring, fin)	0.8710	1.3219	0.9212

4.1.2 POC-English without ambiguity

Table 4.5 contains the evaluations performed on POC-English with no ambiguity. Because this corpus has simple but standard English, in this case we did generate both comparison standards: GS and SS. The first two table rows are helpful as a guide to how similar these standards are to each other. As we can see, their 96% match in both cases shows that there isn't much difference between them, and we can actually notice that all the following rows show a similar score against both standards, as expected. The random baseline stays near 50%, but the sequential one drops substantially, since the parse trees are more complex for this corpus. The OpenCog results all surpass the random baseline, and portray Clique-WIN-dist as the best counting method, followed by LG-Any. In this case, FMI-dist makes a difference only for Clique-WIN-dist. A relevant aspect to highlight is that only the best OpenCog result (Clique-WIN-dist with FMI-dist) performs better or equal to the sequential parsing, while Clique-WIN stays well below it.

Once more, let's take a look at some examples to build upon the previous analysis. First of all, now that we have two standards, we take an example in which the output

Table 4.5: Parsing results for POC-English without ambiguity.

Parsing method	Counting mode	Scoring function	F_1 Gold[%]	F_1 Silver[%]
Manual (GS)	–	–	100	96
LG-English (SS)	–	–	96	100
Random	–	–	46	49
Sequential	–	–	65	69
OpenCog	Clique-WIN-dist	FMI-dist	67	69
OpenCog	Clique-WIN-dist	FMI	63	63
OpenCog	Clique-WIN	FMI-dist	49	47
OpenCog	Clique-WIN	FMI	49	47
OpenCog	LG-Any	FMI-dist	62	62
OpenCog	LG-Any	FMI	62	62

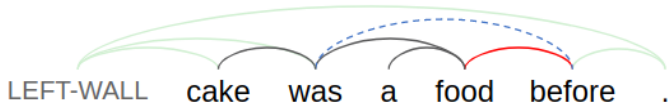


Figure 4.4: Example of an incorrect parse in standard. Corpus: POC-English-NoAmb. Method: LG-English (SS) against GS.

parses for these parsers differ. Below is the LG-English parse output for the sentence - *cake was a food before .* -. This parse is illustrated in comparison to GS in figure 4.4. As one can see from this example, the difference between the two standards appears to be that LG does not seem to be able to manage certain links involving time adverbs correctly. We can also see from this example that this corpora is of higher complexity. The presence of articles in it, for instance, break the sequential nature of the parses for simple sentences because there is a need to connect the verbs directly with the objects and this is not possible without skipping their articles.

```
cake was a food before .
0 ###LEFT-WALL### 1 cake
0 ###LEFT-WALL### 2 was
0 ###LEFT-WALL### 6 .
1 cake 2 was
2 was 4 food
3 a 4 food
4 food 5 before
```

To continue, if we look at sentences that are parsed correctly only by the Clique-WIN-dist with FMI-dist method, we can see that the success of this method lies in the fact that double accounting for distance (in counting and in scoring) is forcing the parser to generate shorter links, causing it to spot correctly the right article-noun links. Given the repetitive appearance of the article 'a' in the text, the other methods do recognize the high FMI vinculating it with its accompanying nouns, but in the moment of parsing, if there is more than one article 'a' in the sentence, only one distance accounting is not enough to break the elevated FMI linkages among

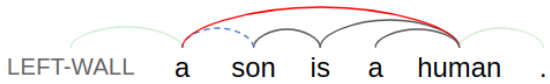


Figure 4.5: Example of a parse with incorrect linking of articles. Corpus: POC-English-NoAmb. Method: OpenCog Clique-WIN + FMI against GS.

far apart article-noun pairs. For example, the following is the output parse for the sentence - *a son is a human .* - with the Clique-WIN + FMI method, which is also illustrated in figure 4.5:

```
a son is a human .
0 ###LEFT-WALL### 1 a
1 a 5 human
2 son 3 is
3 is 5 human
4 a 5 human
5 human 6 .
```

The last example exposes how again in this corpus forcing closer links (i.e. the link among (a, son) instead of $(a, human)$) is indispensable for correct parsing. Yet, in some cases this is not a favorable property. Taking a look at the incorrectly parsed sentences (in all methods) we see that most of the errors for this corpora arise from the reiterative linking of 'is a' and 'was a' instead of the corresponding 'is/was + object' link. If we observe the FMI of these pairs, which are given in table 4.6, we can see that the methods that account for distance overrate the lexical attraction between these pairs. Clique-WIN is the one overrating them less, but this method in turn overestimates the lexical attraction between pairs of nouns like $(mom, cake)$ and $(dad, parent)$, which explains its poorer performance against the other two. The case of LG-Any is one between the other two methods. It appears to have as many problems with article-noun links as the Clique-WIN method but seems to perform better when breaking noun-noun links.

Table 4.6: Example of pair FMI calculation in POC-Turtle: verb + article.

Word Pair	Clique-WIN-dist	Clique-WIN	LG-Any
(is, a)	1.6411	0.9917	1.8173
(was, a)	1.2772	0.6836	1.4379

Finally, we have that there is an inverse effect with distance accounting and the management of negation particles and time adverbs. On one hand, the Clique-WIN method seems to manage better the verb-adverb connections by not breaking them; but it performs very poorly when there are negative particles involved in the sentence, such as in the sentence - *dad was not a parent before .* -. The parse for this sentence is shown bellow and illustrated against GS in figure 4.6. As we can see in the example, this is the reason for this method to evaluate better in GS. On the other hand, Clique-WIN-dist manages negative particles better but does not manage

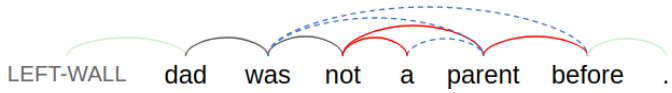


Figure 4.6: Example of a parse with incorrect linking of negation particles and time adverbs. Corpus: POC-English-NoAmb. Method: OpenCog Clique-WIN + FMI against GS.

verb-adverb links so well, thus scoring slightly better against SS. In general it seem like the more complex the structure, the harder it is to find a right balance between not overestimating pair FMI and accounting for distance.

```
dad was not a parent before .
0 ###LEFT-WALL### 1 dad
1 dad 2 was
2 was 3 not
3 not 4 a
3 not 5 parent
5 parent 6 before
6 before 7 .
```

4.1.3 POC-English with ambiguity

We continue with the results from the evaluation of the POC-English with ambiguity corpora displayed in Table 4.7. Once more, GS and SS contain almost-matching parses, earning confidence to the job done by the LG-English parser to produce the SS. The results of sequential and random baselines are quite similar to the corpus without ambiguity, which is expected since the sentence structure is quite similar. However, when it comes to OpenCog results, most experiments increase their F_1 score against both standards, and Clique-WIN with FMI-dist provide the leading parses, even though the other methods don't lag much behind. In fact, almost all OpenCog parses surpass both baselines for this corpus. Our hypothesis towards these results is that when the sentence structure is more complex, there is less need to favor shorter links and thus the positive and negative effects of accounting for distance tend to cancel each other out. This could also be an effect of training with a bigger corpus. We will withal take a look at some examples.

When we take a look at some parses from the LG-English parser (SS) we notice that the verb-adverb management problem remains. Additionally, now we have some special cases where this parser outputs linkages that tolerate cycles. This is certainly not allowed in the GS, and neither it is in any of the OpenCog methods. This creates a situation where some parses score better against SS (like before because of verb-adverb links) and some parses score better against SG because they lack some of the links in the cycle. This causes fluctuations in the overall evaluation, and explains why some methods score slightly better against GS, while other perform better in comparison to SS. To illustrate, take the parse for the sentences - *dad*

Table 4.7: Parsing results for POC-English with ambiguity.

Parsing method	Counting mode	Scoring function	F_1 Gold[%]	F_1 Silver[%]
Manual (GS)	–	–	100	97
LG-English (SS)	–	–	97	100
Random	–	–	46	47
Sequential	–	–	67	68
OpenCog	Clique-WIN-dist	FMI-dist	68	67
OpenCog	Clique-WIN-dist	FMI	68	69
OpenCog	Clique-WIN	FMI-dist	71	72
OpenCog	Clique-WIN	FMI	60	61
OpenCog	LG-Any	FMI-dist	69	68
OpenCog	LG-Any	FMI	69	70

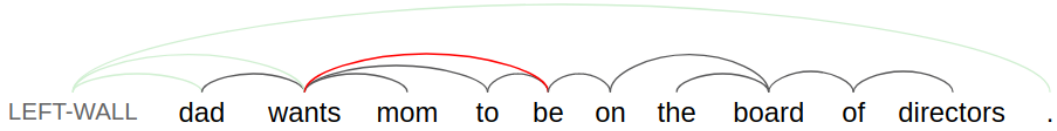


Figure 4.7: Example of a parse with compound verbs. Corpus: POC-English-Amb. Method: LG-English (SS) against GS.

wants mom to be on the board of directors . - and - *mom saw dad with binoculars .* - shown in code next and also in figures 4.7 and 4.8 respectively.

```
dad wants mom to be on the board of directors .
0 ###LEFT-WALL### 1 dad
0 ###LEFT-WALL### 2 wants
0 ###LEFT-WALL### 11 .
1 dad 2 wants
2 wants 3 mom
2 wants 4 to
2 wants 5 be
4 to 5 be
5 be 6 on
6 on 8 board
7 the 8 board
8 board 9 of
9 of 10 directors
```

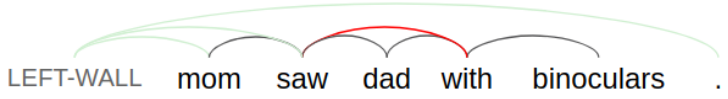


Figure 4.8: Example of a parse with syntactic ambiguity. Corpus: POC-English-Amb. Method: LG-English (SS) against GS.

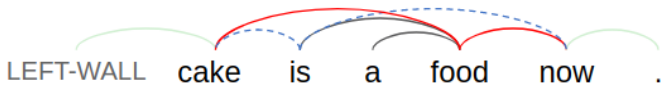


Figure 4.9: Example of an incorrect parse. Corpus: POC-English-Amb. Method: OpenCog Clique-WIN + FMI against GS.

```
mom saw dad with binoculars .
0 ###LEFT-WALL### 1 mom
0 ###LEFT-WALL### 2 saw
0 ###LEFT-WALL### 6 .
1 mom 2 saw
2 saw 3 dad
2 saw 4 with
4 with 5 binoculars
```

The first is an example where the use of compound verbs, *want to be*, makes it unclear where to place the links. The second is an example of a sentence with syntactic ambiguity, where without any further context it is not possible to know whether it is *mom* or *dad* who has the *binoculars*.

Additionally, in the table we notice that for both Clique-WIN-dist and LG-Any counting methods there is no big difference between the scoring function used. The only result that stands out in this respect, is the difference between using FMI-dist or FMI scoring function for the Clique-WIN counting method. When observing example parses, we get the impression that not accounting for distance at all (not in counting and not in scoring) causes the words that do not occur much in the corpus to be connected to everything in the parses. This makes sense, since we know that words that are less frequent carry more information content, and if the link lengths are not considered, links to these words will always be preferred. At last, an example of such word is *food* in the following parse:

```
cake is a food now .
0 ###LEFT-WALL### 1 sausage
1 cake 4 food
2 is 4 food
3 a 4 food
4 food 5 now
5 now 6 .
```

This parse is also illustrated in figure 4.9 against the GS.

4.1.4 Child Directed Speech

Next in complexity is the Child Directed Speech (CDS) corpus, containing adult utterances directed towards young children. Since this corpus is too large to parse manually, we limited ourselves to generate solely a Silver Standard. Even if not all

sentences in the corpus follow a formal English grammar, we assessed by randomly taking a small subset of the parsing results and evaluating their quality manually, that LG-English does an acceptable job at parsing them. However, it's worth noticing that this is the first corpus where the LG returns parses where some words are ignored, since it could not find appropriate parses using them. This implies, that the SS will be missing links involving those word. In contrast, the MST parses always include all words in the sentence, and thus there will be an inevitable mismatch between some links in the OpenCog parses and the ones in our SS, making a 100% F_1 score for the OpenCog parses impossible to achieve.

Table 4.8 shows the summary of the parse evaluations for the CDS corpus. Both the random and sequential baselines perform better than in the previous POC-English corpora, implying that the sentence structure is more linear in this case. The best results among the OpenCog experiments (achieved by both the Clique-WIN-dist and the LG-any methods with FMI-dist) stay short of the sequential baseline, but above the random one. It is noticeable that even the worst OpenCog result (Clique-WIN with FMI) is not far behind the rest. As a matter of fact, if one takes a look at some of the parses output by the different methods, it is straightforward to see that they are very similar. Despite, we will proceed with some representative examples that depict the latest analysis.

Table 4.8: Parsing results for Child Directed Speech.

Parsing method	Counting mode	Scoring function	F_1 Silver[%]
LG-English (SS)	–	–	100
Random	–	–	59
Sequential	–	–	74
OpenCog	Clique-WIN-dist	FMI-dist	68
OpenCog	Clique-WIN-dist	FMI	67
OpenCog	Clique-WIN	FMI-dist	65
OpenCog	Clique-WIN	FMI	63
OpenCog	LG-any	FMI-dist	68
OpenCog	LG-any	FMI	66

The parse for the sentence - *what's Morgie gonna do* - is a perfect example to show the complications we have with our SS for this corpus. The output code of the LG-English parser for this sentence is shown below, followed by the output of all of the OpenCog methods for the same sentence. These two parses are also illustrated in figure 4.10 in the same order:

LG-English:

```
what 's Morgie [gonna] [do]
0 ###LEFT-WALL### 1 what
0 ###LEFT-WALL### 2 's
1 what 2 's
2 's 3 Morgie
```

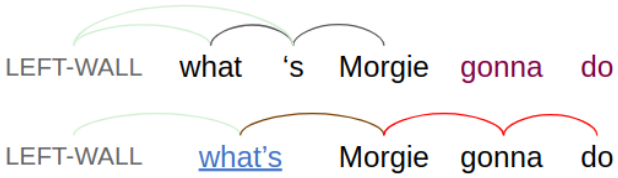


Figure 4.10: Example of a parse excluding words and separating verbs with apostrophes. Corpus: CDS. Method: OpenCog (all 6) against LG-English (SS).

OpenCog:

```
what's morgie gonna do
0 ###LEFT-WALL### 1 what's
1 what's 2 morgie
2 morgie 3 gonna
3 gonna 4 do
```

This example serves to look at two particular issues. The first one is, as mentioned before, that the LG-English parser leaves in some cases a few words out of the parse, which causes our method to have links in its output that don't appear in the standard. In this sentence, the words left out by LG are *gonna* and *do*, which appear in between brackets in the code and in red-wine-colored font in the figure. If we look at the second parse, it is intuitive to see that our method linked at least two of these words correctly: *gonna* and *do*. However, when compared against the standard the link connecting these words will be considered wrong (that is why it is in red in the figure), thus yielding an incorrect and lower evaluation for the parse. This leaves us with the question of weather our standard is appropriate for the task in hand.

The second issue is that LG has the property of being able to separate verbs that are joined to other words through apostrophes, characteristic that our methods do not posses, which in turn also affect the evaluation results. In the previous example, the token *'s* which stands for the verb *is*, is taken separately from *what* for the analysis. On the contrary, the Opencog parsers consider the token *what's*, which is shown underlined and with blue font in the figure, as just one word. As a result, the link connecting this word with the rest of the sentence is also considered incorrect when evaluating, despite being right. Notice that to differentiate this situation in the figure, we show this link in brown instead of red. Conversely, this might explain why all the methods have a similar performance and are all below the sequential baseline.

A somewhat better example of this second issue is portrayed in the parses for the sentence -*that's okay because I'm gonna eat my little blue doughnut*-. The output code of the LG-English parser for this sentence is shown below, followed by the output for the same sentence from the Clique-WIN-dist counting with FMI-dist scoring method. The two parses are illustrated in figure 4.11 in the same order. In this sentence there are two occasions where LG separates the verb from the previous word: it isolated *'s*, which stands for the verb *is*, from the word *that*, and it isolated *'m*, which stands for the verb *am*, from the word *I*. Notice also that LG is allowing

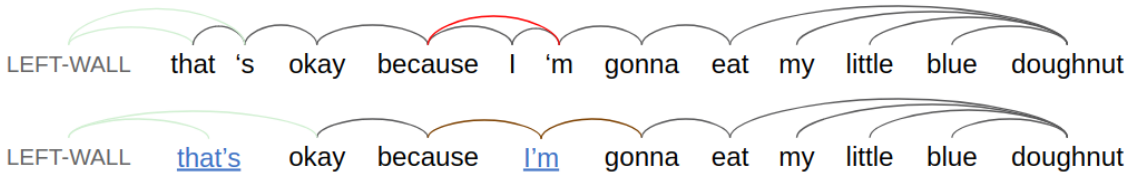


Figure 4.11: Example of a parse that doesn't process apostrophes. Corpus: CDS. Method: OpenCog Clique-WIN-dist + FMI-dist against LG-English (SS).

the use of cycles again to manage this separations[1]. If we look at the second parse we see that, except for the first word, our method did quite a good job at parsing this sentence, which is not a simple one. Howbeit, when evaluating, because of this different tokenization of the methods, two links (the ones in brown) are perjudicially considered wrong.

LG-English:

```
that 's okay because I 'm gonna eat my little blue doughnut
0 ###LEFT-WALL### 1 that
0 ###LEFT-WALL### 2 's
1 that 2 's
2 's 3 okay
3 okay 4 because
4 because 5 I
4 because 6 'm
5 I 6 'm
6 'm 7 gonna
7 gonna 8 eat
8 eat 12 doughnut
9 my 12 doughnut
10 little 12 doughnut
11 blue 12 doughnut
```

Clique-WIN-dist + FMI-dist:

```
that's okay because i'm gonna eat my little blue doughnut
0 ###LEFT-WALL### 1 that's
0 ###LEFT-WALL### 2 okay
2 okay 3 because
3 because 4 i'm
4 i'm 5 gonna
5 gonna 6 eat
6 eat 10 doughnut
7 my 10 doughnut
8 little 10 doughnut
9 blue 10 doughnut
```

[1] Even though there is no GS to compare the SS against it, we still marked in red the edge that would be considered as extra by a manual annotation, just to highlight the issue.

4.1.5 Gutenberg Children

The Gutenberg Children corpus increases substantially the language complexity, using proper English phrases with sometimes complicated structure, but using a limited vocabulary and content typical in Children literature. Among its peculiarities, it's relevant to point out that this corpus commonly uses comma separations or inline quotations to clarify an aspect or to specify what some character in the story said, as we will show in some examples later. This kind of in-lines are sentences by themselves, independent from the sentence containing them, so statistics for words near the transition to/from a quotation will get some amount of noise from neighbors that do not belong to the same syntactic sentence. The possible solution of separating quotations into their own sentences is complicated to automate and would probably require a good amount of language-dependent rules, without mentioning that the sentence containing the in-line might result incomplete without it. Consequently, we decided to acknowledge this problem at the moment of analyzing, but ignore it when evaluating hoping to see if the obtained metrics reflect or not this phenomena.

Moreover, due to the computational effort involved in parsing this corpus, we decided to do it only with a couple of the most successful methods from our previous experiments. After reviewing the results obtained from parsing the already analyzed corpora, we chose Clique-WIN-dist + FMI-dist and LG-any + FMI-dist as the two parameter combinations to use for the subsequent parsings. The results obtained from parsing the Gutenberg Children corpus are shown in Table 4.9.

Table 4.9: Parsing results for Gutenberg Children.

Parsing method	Counting mode	Scoring function	F_1 Silver[%]
LG-English (SS)	–	–	100
Random	–	–	41
Sequential	–	–	61
OpenCog	Clique-WIN-dist	FMI-dist	48
OpenCog	LG-Any	FMI-dist	52

Looking at the table, we observe that both the sequential and random baselines are the lowest we have seen for any corpora so far, which could reflect the fact that more words are ignored by the SS than before. A lower sequential baseline could also imply a less linear structure in the sentences, as we would expect given the peculiarity of the text. We observe an equivalent drop in the F_1 scores for the outputs of the two method combinations used, but they remain in between both baselines. The fact that all scores drop (both for baseline and OpenCog methods) indicates a larger difference between our standard and the rest of the parses; which could also be explained with the aforementioned increase in words ignored by the LG-English parser. On the other hand, this decline could also be attributed to the elevated complexity level of the structures and the stretched lengths of the sentences introduced by this corpus. To clear the doubt, we will take once more a look at some

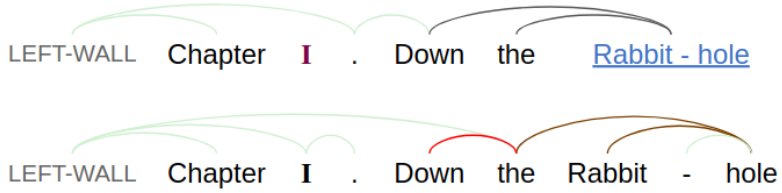


Figure 4.12: Example of a parse with different tokenization. Corpus: Gutenberg Children. Method: OpenCog LG-ANY + FMI-dist against LG-English (SS).

representative examples.

The following is an example of a parse where LG ignored one of the words. In this case the sentence being parsed is not a grammatical sentence per se, but the tittle of a book chapter. The output parse of the LG-English parser for this 'sentence' is shown in code below, followed by the output of all of the LG-Any + FMI-dist method for the same sentence. These two parses are also illustrated in the same order in figure 4.12. The word that LG leaves out of the parse is the roman numeral I, shown in red-wine-colored font in the figure, and representing the number of the chapter. One interesting thing that happens in this sentence, is that both methods recognize the need to separate *Chapter I.* from the actual name of the chapter, and they do so by making use of the punctuation and left-wall. Given that punctuation and left-wall links are ignored in the evaluation, this is not an example where these aspects affect the metric obtained. However, this parse does exemplify another issue that does alter the evaluation, which we will analyze next.

LG-English (SS):

```
CHAPTER [I] . down the Rabbit-Hole
0 ###LEFT-WALL### 1 CHAPTER
0 ###LEFT-WALL### 3 .
3 . 4 down
4 down 6 Rabbit-Hole
5 the 6 Rabbit-Hole
```

LG-Any (OpenCog):

```
chapter i. down the rabbit-hole
0 ###LEFT-WALL### 5 the
0 ###LEFT-WALL### 1 chapter
0 ###LEFT-WALL### 2 i
2 i 3 .
4 down 5 the
5 the 8 hole
6 rabbit 8 hole
7 - 8 hole
```

Just like with the apostrophes in the CDS corpus, we have a case here where the LG parser uses a different tokenization than OpenCog, which is harming the evaluations.

It is now the OpenCog method the one that considers more words in the sentence. In this occasion, the difference lies in the interpretation of the hyphen between *Rabbit* and *Hole*: LG interprets the hyphen as serving the function of connecting the two words as one, while OpenCog just interprets it as any other punctuation mark. If we look at the links for the second part of the sentence (the title of the chapter) all of the links that matter are considered as wrong, even though intuitively some of them make sense. In the figure, we drew the arcs for this links in brown and not red to distinguish them from the ones that, in a standard with consistent tokenization, would actually be considered right.

The second example we show is one where the use of both quotations and a semicolon combine two separate sentences together into one. Once again we show the code of the output parse of the LG-English parser for such example followed by one from an OpenCog method, save that it is from the Clique-WIN-dist + FMI-dist method this time. These two parses are illustrated following the same order in figure 4.13. As we can see from the red-colored font in the figure, even though there are tokens that LG is ignoring in this parse, none of them are actual words. This example actually serves to highlight two things. The first one is that LG does a very good job at parsing complex sentences like this one. Therefore, despite the evident problems we have found with this standard and even though the numerical scores might not be statistically reliable for conclusions, this parser does work as a good referent to compare among the different methods.

LG-English (SS):

```
['] What a curious feeling [!] ['] said Alice ; ['] I must be
    shutting up like a telescope . ']
0 ###LEFT-WALL### 5 feeling
0 ###LEFT-WALL### 8 said
0 ###LEFT-WALL### 10 ;
0 ###LEFT-WALL### 20 .
2 What 5 feeling
3 a 4 curious
3 a 5 feeling
4 curious 5 feeling
5 feeling 8 said
8 said 9 Alice
10 ; 12 I
10 ; 14 be
12 I 13 must
13 must 14 be
14 be 15 shutting
15 shutting 16 up
15 shutting 17 like
17 like 19 telescope
18 a 19 telescope
```

Clique-WIN-dist (OpenCog):

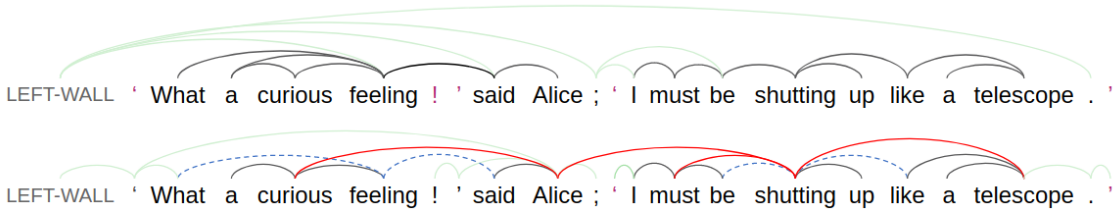


Figure 4.13: Example of a parse with inline quotations. Corpus: Gutenberg Children. Method: OpenCog Clique-WIN-dist + FMI-dist against LG-English (SS).

```
' what a curious feeling ! ' said alice ; ' i must be shutting
    up like a telescope. '
0 ###LEFT-WALL### 1 '
1 ' 9 alice
1 ' 2 what
3 a 4 curious
4 curious 9 alice
4 curious 5 feeling
6 ! 7 '
7 ' 9 alice
8 said 9 alice
9 alice 15 shutting
9 alice 10 ;
11 ' 12 i
12 i 13 must
13 must 15 shutting
13 must 14 be
15 shutting 19 telescope
15 shutting 16 up
17 like 19 telescope
18 a 19 telescope
19 telescope 20 .
20 . 21 '
```

Finally, the second and last observation we get from this example is that it is actually true that having independent sentences joined together into one generates some noise in our methods. This is easily observable in the two incorrect and long links: (*curious*, *Alice*) and (*Alice*, *shutting*), probably generated by a high FMI. It seems as if the parser was trying to connect the main subject, *Alice*, to its adjective and verb; except that these two last are really part of other sentences and therefore complementing other nouns: $feeling$ and I. This can give the impression that the way Clique-WIN-dist accounts for distance is a better alternative to LG-Any, but it is actually the opposite. Clique-WIN-dist counts frequencies only for words within the window, 6 in our case (see section 3.2). In this example, both links are within this window, so the effect is the same for both methods; yet, for other sentences in which the separate sentence is in the middle, using a relatively small window when

counting makes it practically impossible for our parser to link the two separated parts of the sentence. LG-Any, on the other hand, does not favor many long links because the natural probability of such links being part of a random parse is low; however, it is not zero, allowing for exceptions when necessary. Indeed, this might explain the slightly better performance of the LG-Any counting method when it comes to more complex texts such as Gutenberg Children.

4.1.6 Brown

The Brown corpus contains the most complex, adult-style English that we analyze for this thesis project. Like the Gutenberg Children corpus, it contains complex sentence structure and frequent inline quotations. An extra characteristic of this corpus is that it is composed of text coming from different sources and encompassing different genres: News, Literature, Religious texts, etc.; so the vocabulary is not consistent in different parts of the corpus. It represents a wider variety of the English language, at least of the one used in 1960's texts, so it represents a good challenge for a language learning algorithm. Given the results we obtained from the last corpus, we decided to maintain the two methods used before, but add one more to analyze not only the effect of closer word counting in compound sentences, but also of distance accounting when parsing. Thus, we additionally used the LG-Any with FMI to parse this corpus.

The results obtained after parsing the Brown corpus are shown in Table 4.10. Interestingly, the results are quite similar to those of the previous corpus: lower baseline results than for the simpler corpora, and OpenCog results that lie in between those two (although reaching a bit lower F_1 scores this time). Similar baselines likely indicate a similar sentence structure and complexity to Gutenberg Children. On the other hand, after pre-cleaning, the Brown corpus contains a smaller vocabulary than Gutenberg Children (42k vs 54k unique words, respectively), and is as well a much smaller corpus (0.5M vs 2.7M words, approximately). This tells us that the length of the corpus has only a slight effect on the quality of the parses. Also, the variety of topics handled in the Brown corpus does not appear to difficult significantly the task at hand. Last but not least, the marginally higher score we see for FMI-dist against FMI, suggest that accounting for distance in the scoring function does appear to have a positive in the results in moderating the long links allowed by the LG-Any counting mode parser.

Although the evaluation scores speak for themselves, for completeness, we will look at one example here as well. This example compares the result of parsing the sentence - *He will be succeeded by Ivan Allen Jr., who became a candidate in the sept.* - with the next three methods in this order: LG-English (SS), Clique-WIN-dist, and LG-Any (both scoring functions). In this case, because the parses are very long we only provide an illustration of them in figure 4.14, and no code like in the previous examples. In the figure, the first thing we notice is that the method LG-Any allows for longer links; and even though this particular example doesn't depict it, this does account again for its slightly better performance. The second thing we notice is that

Table 4.10: Parsing results for Brown.

Parsing method	Counting mode	Scoring function	F_1 Silver[%]
LG-English (SS)	–	–	100
Random	–	–	41
Sequential	–	–	61
OpenCog	Clique-WIN-dist	FMI-dist	46
OpenCog	LG-any	FMI-dist	48
OpenCog	LG-any	FMI	46

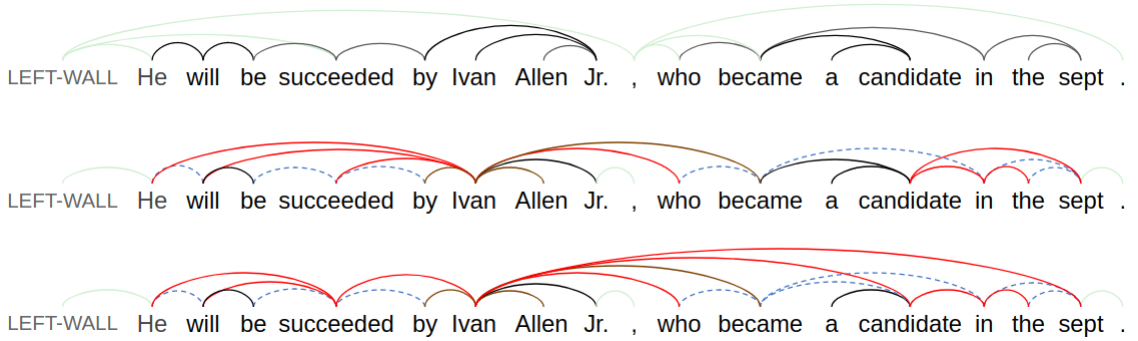


Figure 4.14: Example of a parse with a compound sentence. Corpus: Brown. Method: OpenCog Clique-WIN-dist (2) and LG-Any (3) against SS (1).

our method and the standard seem to disagree in the way of linking proper names, such as *Ivan Allen Jr.*. In this case, LG considers the word *Jr.* to be the main connection, while our parser interprets it to be rather *Ivan*. We considered this kind of differences not critical, despite the fact that they do affect the evaluation score, and thus marked the respective links is brown denoting a mistake of a minor level than the red ones.

Finally, one more thing we can notice is that again LG makes use of punctuation to separate the parses for the two sentence parts. However in this case, this might not be the most appropriate approach, given that the second sentence part relates to the subject of the first one. As a matter of fact, if we take away the walls and the punctuation, the LG resulting parses for the sub-sentences would be totally disconnected; while the OpenCog resulting parses would still be connected by the link $(Ivan, became)$. Once again we considered this link not to be a meaningful error, almost tending to a correct approach, and therefore marked it also in brown in the figure.

4.1.7 Penn Treebank

The last corpus that we analyze is a section of the Penn Treebank. In addition to the dependency parse structures the bank provides, the corpus comes with raw text from WSJ material, which we use as input to our pipeline. An extra benefit of using this corpus in comparison to the previous ones is that it comes with its own Gold Standard. Because of the length of previous corpora, we could only hand-craft a

GS for the smaller ones, and resorted to using only Silver Standards for the more realistic ones. Now, for this corpus we are able to report again the six combinations of parameters. Table 4.11 contains F_1 scores for the parses of the Penn Treebank with our methods against both standards.

Table 4.11: Parsing results for Penn Treebank.

Parsing method	Counting mode	Scoring function	F_1 Gold[%]	F_1 Silver[%]
Penn Treebank (GS)	–	–	100	57
LG-English (SS)	–	–	56	100
Random	–	–	36	40
Sequential	–	–	49	57
OpenCog	Clique-WIN-dist	FMI-dist	38	42
OpenCog	Clique-WIN-dist	FMI	36	40
OpenCog	Clique-WIN	FMI-dist	32	35
OpenCog	Clique-WIN	FMI	29	31
OpenCog	LG-Any	FMI-dist	40	44
OpenCog	LG-Any	FMI	39	42

When observing the numbers in the table, first of all, we notice that, in contrast to the much simpler POC corpora, the difference between GS and SS is quite large: they score a bit above 50% against each other, and not far above the sequential baseline for each case. This situation possibly raises again a question about our SS method, which is leaving a number of words out of its parses, it's using different tokenizations and is perhaps, to some extent, responsible for the decaying parse scores of complex corpora against it. As a counter argument in favor of the SS, the two baseline parses do not perform drastically lower against it for this corpus, while there is definitely a poorer match between the baselines and the GS. Howbeit, when looking at the OpenCog parses, we find that many of the methods don't even perform better than random parsing for any of the standards, and all of them are considerably worse than the sequential. Systematically, the results against GS are lower than against SS. An interpretation of this behavior is that this corpus indeed has more complex structure than the previous ones, as hinted by the poor baselines performance, as well as a lot smaller size (38k words) to gather enough statistics about the underlying grammar.

To support the underlying argument, we provide two last examples. The first example is for the sentence - *It has no bearing on our work force today.* - which is a relatively linear phrase with no tokenization issues nor composition of various sentences. Figure 4.15 illustrates the result of parsing this sentence with the best method, LG-Any with FMI-dist, against the two standards. In the figure, the first parse is the one provided by the treebank, the GS; the second one is the parse output by LG-English, the SS, but compared against GS; and the third on is the resulting parse of applying the selected OpenCog method over the sentence and compared against both standards. The second example is for the sentence - *This time, it was for dinner and dancing – a block away.* -, which is also a relatively linear phrase

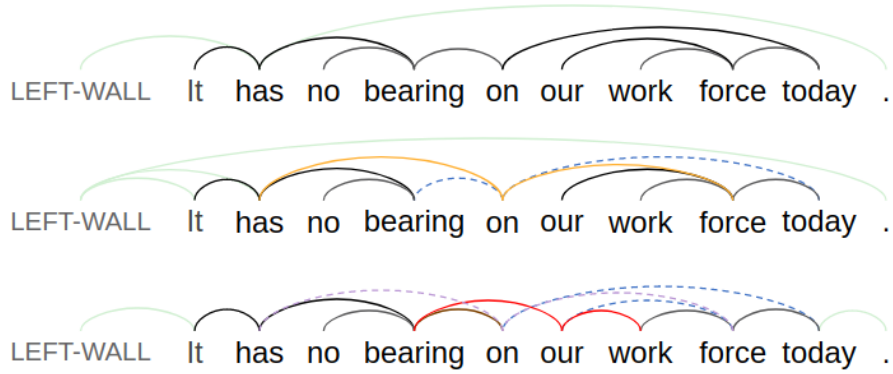


Figure 4.15: Example of a simple parse. Corpus: Penn Treebank. Method: LG-Any with FMI-dist (3) against GS (1) and SS (2).

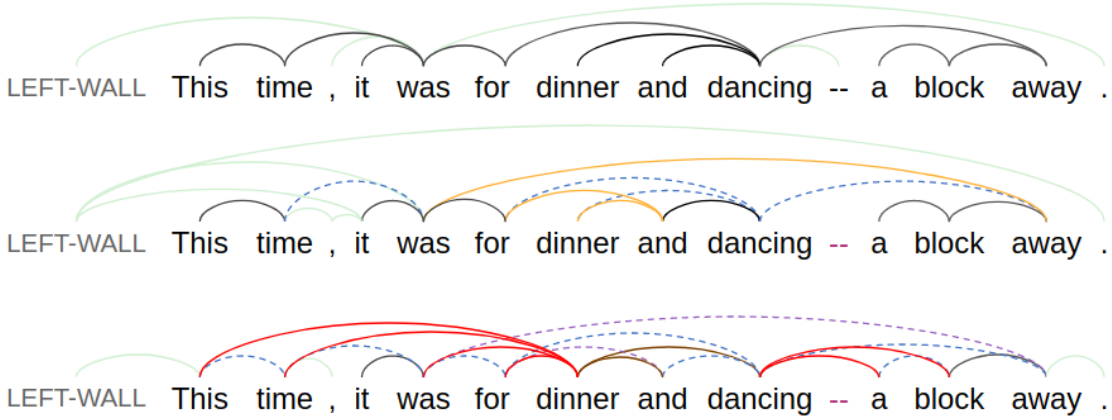


Figure 4.16: Example of a complex parse. Corpus: Penn Treebank. Method: LG-Any with FMI-dist (3) against GS (1) and SS (2).

but also a compound one with the use of some extra punctuation. Figure 4.16 illustrates the result of parsing this second sentence with the same methods as the first example and in the same order.

The coloring scheme used for the links in these last two examples is as follows. Black and green continuous links remain being the correct and ignored ones, respectively. Yellow continuous links for the LG parse (SS) are links that don't appear in the GS. Brown continuous links for the OpenCog parse are links that appear in one of the standards but not on both. Purple dashed arcs for this same parse represent links that appear only in the SS standard but not in GS nor in the parse in mention. Blue dashed arcs in both LG and OpenCog parses represent links that appear in the GS but not in the corresponding parse. Finally, red links for the OpenCog parse are links that do not appear in any of the two standards. In general, the take-away we get from these two examples is that the more complex the structure of the sentence the more the two standards differ, but and also the more deviated our method is from both of them, which is totally consistent with the results obtained in the evaluation.

4.2 Word-sense Disambiguation

After observing results from passing the corpora through our parsing pipeline, we proceeded to add an extra pre-processing step to disambiguate uses of the words in the analyzed texts with the help of AdaGram, and then parse the disambiguated corpora. This would, in principle, allow our statistical parsing method to obtain more fine-grained statistics about word use resulting in better parses, as explained in section 3.3. It's important to have in mind that this strategy would have the effect of gathering less statistics about each disambiguated word, which could potentially impact the parse quality negatively, specially if noise in the data becomes more relevant. For the experiments of this nature, in this thesis we focused on two corpora: POC-English with ambiguity, just for the purpose of having a simple and controlled corpus to experiment with,; and our largest corpus: Gutenberg Children, where the actual performance of the approach will be tested.

4.2.1 WSD in POC-English

As before, we started our learning experiments with the simplest suitable corpus: in this case, POC-English with ambiguity. As specified in section 3.4.2, we trained our model's hyper-parameters for this corpus in a WSD task using a hand-crafted disambiguation Gold Standard (GS)[2], and PWFS and OF as metrics. Given the short size of the corpus, the test set comprises all sentences in the corpus, but only two words are annotated as polysemous ones: 'saw' and 'board', with only two senses each. After initial exploration of the parameter space with HyperStudy, we found out that given the very limited length of the POC-English corpus, we needed to train the models for a long number of epochs to obtain sense vectors that reflected the use of the ambiguous words in the corpus. We decided for 1 million epochs for the subsequent trials. Then we performed a series of experiments where the prior parameters were explored by HyperStudy to train AdaGram models, annotate the original corpus with them, and evaluate their performance using the given metrics. We were interested in finding the relevant hyper-parameters to train an AdaGram model, so we explored those and kept the annotator parameters fixed (meaning, Window-annotate and Min-prob from the parameters listed in section 3.3).

Table 4.12 summarizes the linear effects[3] found for each variable towards the available scores[4]. The linear effects give an idea of how relevant each variable is towards each score, with a higher absolute value meaning more influence, and a negative value implying a negative correlation. Hence, we notice that Window-model is the variable that has a larger influence on the results: in average, it increases PWFS by 15%, but also causes $\sim$ 16% more words to be overdisambiguated; e.g. in the

[2] This standard can be found at `http://88.99.210.144/data/asuarez-disambiguate-pre/EnglishPOC/EnglishPOC.txt_disamb_gold`

[3] To calculate the linear effect of a variable with two values on a certain score, HyperStudy averages all scores when that variable takes each of the values, and reports their difference.

[4] The detailed results of our multiple runs and range of parameters explored can be found in `https://docs.google.com/spreadsheets/d/1CugCrAmgxG2idi2JsLguGd07SUWNCYIYxUMbVpJmxRU/edit?usp=sharing`

POC-English corpus, the word *cake* comes both in the sentence -*A mom likes cake* .- and in -*Cake is a food* .-, which share no context words at all, yet are considered to represent the same meaning. The second largest effects are caused by Remove-top, which has a negative correlation with both scores: the more frequently used words we eliminate from the model, the lower the PWFS score is, and also the more single-sense words are contained in the model.

Table 4.12: Hyper-parameter effects on WSD scores for EnglishPOC.

Hyper-parameter	Effect on PWFS	Effect on OF
Dimensions	0.001	-0.010
Min-freq	0.010	-0.037
Remove-top	-0.130	-0.041
Alpha	0.008	0.024
Window-model	0.148	0.156

Once we detected the variables with largest linear effects, we proceeded to design a couple more fine-grained experiments. With the guidance of HyperStudy, we were able to locate the best performing model among our experiments, and then manually chose annotator parameters to get the best disambiguation results. It is important to note that the intention here was not to assume that the best performing set of parameters will provide us with great models in subsequent corpora. The idea was to test whether AdaGram is actually able to disambiguate word senses based on a very simple, proof of concept corpus, and subsequently test the performance of MST-parsing that disambiguated corpus.

We found two promising models, whose parameters are summarized in Table 4.13. Figure 4.17 shows a histogram on how the two models perform against the GS, with example disambiguated sentences from each of them. We can notice how Model 1[5], even if it does not achieve perfect disambiguation, is able to annotate the corpus properly. Model 2[6], which clearly distinguishes each use of the ambiguous words, also annotates other words which the GS doesn't consider ambiguous. This behaviour is not completely unexpected: given the nature of the corpus, where some supposedly unambiguous words appear in totally different contexts, it is natural to expect some overdisambiguation. What would be finally important to our project is that these annotated corpora can provide better parsing accuracy to the MST pipeline.

We decided to parse the annotated corpus using the best-performing parameter combination in the non-disambiguated exercise. Table 4.14 presents parsing scores against both GS and SS for parses obtained from POC-English annotated with our

[5] Available at: `http://88.99.210.144/data/asuarez-disambiguate-pre/EnglishPOC/best_annotated_270718/EnglishPOC.txtadagramWO8D50E1000000M1A1.3W3R0_disamb`

[6] Available at: `http://88.99.210.144/data/asuarez-disambiguate-pre/EnglishPOC/100disamb_18overdisamb/EnglishPOC.txtadagram_EnglishPOC.txt_WO8D300E1000000M5A1.3W3_disamb`

Table 4.13: Parameters and scores for the best disambiguated POC-English models.

Model	Dim	Min-freq	Rm-top	Alpha	WM	WA	Min-prob	PWFS	OF
1	50	1	0	1.3	3	3	0.33	0.907	0.045
2	300	5	0	1.3	3	3	0.05	1.0	0.409

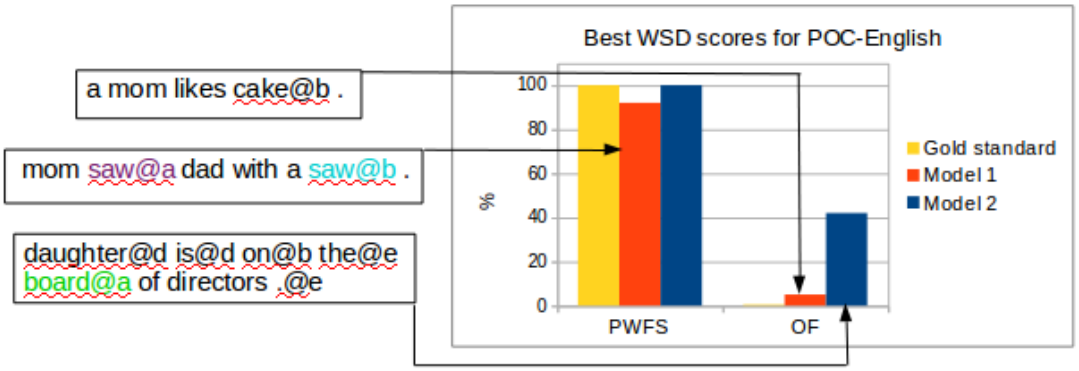


Figure 4.17: Examples and results for Model 1 and 2, trained for WSD in the POC-English corpus. Ambiguous word senses, according to the GS, are colored in the example sentences.

best trained AdaGram models. We show the non-disambiguated results from Table 4.7 as a reference, as well as the standards and baselines. In this case, the best scores for each OpenCog method are shown in blue font. We can notice that for every single OpenCog combination of parameters, the annotated corpora performed better than the unnanotated ones. The best overall parsing score for the corpus (shown in green font) is achieved by Model 1, parsed with LG-Any + FMI.

These improvements, although not dramatic, suggest that using disambiguated word senses actually helps the MI-scores to make more sense for the MST algorithm, presumably because now ambiguous words do not appear in such different contexts when pair-counting, thus the pairs' calculated mutual information contains less noise. It is also relevant to highlight that the high overdisambiguation included in Model 2 does not seem to deteriorate parsing scores; this can be understood if we think that overdisambiguated words are just becoming more fine-grained versions of themselves. In this case, the statistics gathered by each word-sense will be less in quantity but not necessarily in quality, and could still provide the right information to the MST algorithm for linking them correctly.

4.2.2 WSD in Gutenberg Children

After the results shown in the previous subsection, we decided to try to disambiguate a more challenging corpus: Gutenberg Children. In this case, creating a Gold Standard by annotating the whole corpus was not feasible due to the corpus length.

Table 4.14: Parsing results for POC-English after disambiguation.

Parsing method	Dis-ambiguation	Counting mode	Scoring function	F_1 Gold[%]	F_1 Silver[%]
Manual(GS)	–	–	–	100	97
LG-English(SS)	–	–	–	97	100
Random	–	–	–	46	47
Sequential	–	–	–	67	68
OpenCog	–	Clique-WIN-dist	FMI-dist	68	67
OpenCog	Model 1	Clique-WIN-dist	FMI-dist	69	70
OpenCog	Model 2	Clique-WIN-dist	FMI-dist	70	71
OpenCog	–	Clique-WIN	FMI-dist	71	72
OpenCog	Model 1	Clique-WIN	FMI-dist	68	69
OpenCog	Model 2	Clique-WIN	FMI-dist	73	74
OpenCog	–	LG-Any	FMI	69	70
OpenCog	Model 1	LG-Any	FMI	74	75
OpenCog	Model 2	LG-Any	FMI	72	73

To guide the disambiguation parameter search, we considered using a test set built from a subset of the corpus sentences that contain a selection of two ambiguous words, with two senses each: 'face' and 'love', both of which could be used as a noun or as a verb. The idea is the same as before: train a model with the entire corpus, disambiguate the corpus using the model, tune the training hyper-parameters by measuring the $PWFS$ against the disambiguation GS[7]. We performed a similar parameter-space search as for POC-English by using HyperStudy and the same metrics. Finally, in this case we can evaluate the quality of our model by using the SCWS database.

The way we proceeded is as follows. First, we ran an experiment just varying the number of training epochs, and noticed that a larger number of epochs didn't seem to improve the model, which is easily explainable given that this corpus is considerably bigger; so we decided to stick to shorter runs than in the previous corpus. Then we set a second experiment to explore which of the model training hyper-parameters have a larger effect on the results, by fixing the annotator parameters. We set window-annotate to the same value as window-model and, since we noticed in the last subsection that overdisambiguation didn't seem to have a considerable negative effect, we decide to set min-prob to a low value (0.05) to try to catch the disambiguation potential of the model, regardless of overdisambiguation.

The linear effect plots for $PWFS$ obtained are shown in Figure 4.18. Each plot in the figure displays the $PWFS$ score for all models trained in the experiment at a given value of each studied parameter; hence, the top-left box displays the linear effect that the number of dimensions of the model has on the score, etc. We can

[7] I.e. the test set manually disambiguated for the two target words, which is available at: `http://langlearn.singularitynet.io/data/asuarez-disambiguate-pre/Children_Gutenberg/annotated_GC_dotless_MP005W2/`

appreciate that Window-model is the most-affecting parameter to $PWFS$, and that it has a negative correlation: the larger the context window while training, the worst the model scores, in average. On the contrary, both Dimensions and Min-freq have a positive linear effect.

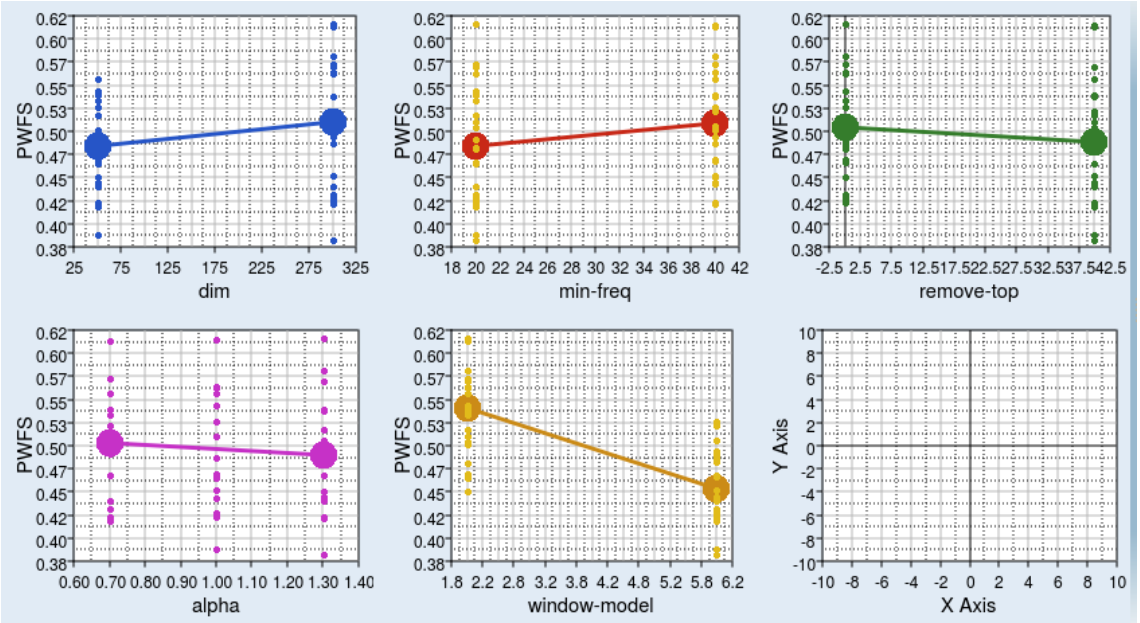


Figure 4.18: Linear effects for selected hyper-parameters on $PWFS$ score for the Gutenberg Children corpus.

Guided by the learned model behavior, we set the range for the hyper-parameters for a Genetic Algorithm Optimization using HyperStudy. Details of the algorithm can be reviewed in the software's user manual[8] or in a book on stochastic optmization methods [50]. In brief, what this optimization will do is train a number of models with random values for the hyper-parameters (within the allowed ranges), evaluate each model to asses its performance, create new models by combining parameters from the top models from the past generation and random modifications, and repeat the cycle until error tolerance is reached. After the optimization finished, the algorithm returns a combination of parameters that achieve the best results. The ones we obtained for our experiment are given in Table 4.15. It is important to notice that the disambiguation results are not close to perfect: although OF remains quite low, the $PWFS$ score implies that not all instances of the GS ambiguous words were correctly annotated. The situation is understandable in a realistic corpus like this one, since words might appear in unique contexts sometimes and senses can be subtle in others.

After tuning the hyper-parameters of the model, we proceed to evaluate it intrinsically using the the SCWS database detailed in section 3.3, which is completely independent from the data used to train the model. The procedure for evaluation

[8] `https://connect.altair.com/CP/kb-view.html?f=2&kb=180869`

Table 4.15: Parameters and scores for the best disambiguated Gutenberg Children model.

Dim	Epochs	Min-freq	Rm-top	Alpha	WM	WA	Min-prob	PWFS	OF
300	80	38	80	1	2	2	0.05	0.772	0.02

was detailed in Section 3.4.2, and consists in calculating four similarity measures (two of which take context into account) for the word pairs in the dataset that are contained in the model. Table 4.16 contains Spearman's ρ coefficients for the different scores of our top-performing model against SCWS, as well as scores reported by Huang and colleagues [27, 16]. It's important to highlight that those models are not trained in the same corpus and the comparison may not be fair[9], but we show them nonetheless as a reference only, to realize how well our model builds its representations.

Table 4.16: Spearman's ρ correlation for different similarity scores for our best disambiguated Gutenberg Children model. Results for all models but Model 1 are taken from Huang et al. [27]

	Nbr. pairs	AvgSim	MaxSim	AvgSimC	MaxSimC
C&W-S	?	0.57	0.57	–	–
Model-S	?	0.586	0.586	–	–
Model-M	?	0.628	–	0.657	–
Model 1	76	0.646	0.647	0.184	0.596

The first column, 'Nbr. pairs', indicates how many of the word pairs in SCWS were evaluated for the model, which was trained in Gutenberg Children and does not contain many of the words in the word similarity database. We can see that only 76 out of 2003 word pairs were used for evaluation, which is understandable given the nature of our Children's books corpus and its limited vocabulary. Contrarily, the SCWS corpus contains words obtained from more varied text, like Wikipedia [27]. At the same time, a similar problem occurs with the contexts used to disambiguate the word senses: even when a word in the word pair does exist in our model, it is likely that many of the words in the context window will not be available to use and a majority of functional words (as opposed to content words) will be used to disambiguate, making that task difficult to excel.

As an example, consider the word pair 'baby' and 'mother' in the SCWS dataset. The word 'baby' comes in the following context:

> Lysergic acid is made by alkaline hydrolysis of lysergamides like ergotamine , a substance derived from the ergot fungus on rye , or from ergine (lysergic acid amide , LSA) , a compound that is found in morning glory (Ipomoea tricolor) and hawaiian baby woodrose (Argyreia

[9] For instance, Huang et al. used a billion-word snapshot of Wikipedia to train their model.

> nervosa) seeds . LSD is a chiral compound with two stereocenters at the carbon atoms C-5 and C-8 , so that theoretically four different optical isomers of LSD could exist . LSD , also called (+) - D-LSD , has the

From these, the words that are contained in our best model and included within the window for the pre-disambiguation step are: ‘)’, ‘(’, ‘morning’, ‘found’, ‘(’, ‘)’, ‘two’, and ‘four’. None of these are very indicative of the sense in which the ‘baby’ is used in the test, and it’s expected that the contextual similarity metrics (AvgSimC and MaxSimC) will perform well.

Correspondingly, the word ‘mother’ comes in the context:

> Meanwhile , the truth about Willoughby ’s real character starts to emerge ; Colonel Brandon tells Elinor that Willoughby had seduced Brandon ’s ward , fifteen-year-old Eliza Williams , and abandoned her when she became pregnant . Brandon was once in love with Miss Williams ’ mother , a woman who resembled Marianne and whose life was destroyed by an unhappy arranged marriage to the Colonel ’s brother . Fanny Dashwood , who is also in London for the season , declines her husband ’s offer to invite the Dashwood girls to stay with her .

and the words that can be used to disambiguate it’s sense are: ‘miss’, ‘love’, ‘one’, ‘real’, ‘woman’, ‘whose’, ‘life’, and ‘an’. A similar situation is observed in this case, where only a limited number of words hint towards the sense implied in the example.

This situation helps to understand why the coefficients for AvgSimC and MaxSimC are considerably lower than the rest. On the other hand, scores that do not depend on the context, but only on the learned word sense representations (AvgSim and MaxSim) perform comparably well in relation to the cited references. We can presume that, within the limited vocabulary we used for training, the vectors learned by our model can compare to other published models. However, when it comes down to distinguishing context, unfortunately we are not in a position to conclude anything.

The last evaluation we perform is an extrinsic one: does the parse quality of the Gutenberg Children corpus increase when we annotate it with senses from our top-performer embeddings model? The answer to this question lies in table 4.17, which includes the baselines and previous OpenCog results for comparison. We parse the annotated Gutenberg Children corpus[10] using the same parameter combinations used for parsing before disambiguation.

We notice that the results do not show a dramatic change from the previous scores. In particular, the parses from the Clique-WIN-dist method worsened by 1%, while the LG-any produced a slightly better parsed corpus after annotation. Both the

[10] Available at: `http://langlearn.singularitynet.io/data/asuarez-disambiguate-pre/Children_Gutenberg/annotated_GC_dotless_MP005W2/`

PWFS and the SCWS evaluation were telling us that the model does not contain perfectly disambiguated word embeddings, so it can be understood that the parses did not increase in quality with this input.

Table 4.17: Parsing results for Gutenberg Children after disambiguation.

Parsing method	Disambiguation	Counting mode	Scoring function	F_1 Silver[%]
LG-English (SS)	–	–	–	100
Random	–	–	–	41
Sequential	–	–	–	61
OpenCog	–	Clique-WIN-dist	FMI-dist	48
OpenCog	Model 1	Clique-WIN-dist	FMI-dist	47
OpenCog	–	LG-Any	FMI	50
OpenCog	Model 1	LG-Any	FMI	52

From these and the previous POC-English disambiguation efforts, one can infer that only when the words in a corpus are disambiguated properly, the parsing algorithm can benefit from the fine-grained information carried by the separated word senses. On the contrary, if the different word prototypes cannot distinguish clearly a word use in its context, then annotating them incorrectly doesn't provide a better input to the parsing process.

5
Conclusions

In this thesis we implemented and tested six different methods for an unsupervised parser. Our parser consisted of three main stages: gathering of word pair statistics from unannotated text, estimation of mutual information scores for such word pairs, and using the scores obtained in an MST algorithm to parse the sentences from the text. The six methods were a combination of three different different ways of gathering the word pair statistics, and two different scoring functions used in the MST algorithm. To evaluate the output of our method we used two standards to compare against: a manually crafted goal and a supervised parser; and two guidelines to use as reference: random and sequential parsing. The metric we used was F_1, which is commonly used in classification tasks.

Furthermore, we also explored how the quality of the input we feed into our methods affect the output of them. We did this by feeding the parser and analyzing corpora of different size and complexity, but we also experimented with the ambiguity present in a text. We first trained AdaGram models to learn vector representations of words from our ambiguous corpora, and selected the one that best disambiguated our test sentences. Then we assessed the quality of the model by comparing against the SCWS dataset similarity of word senses or meanings within contexts, using the pairwise f-score. Once we determined the model was good enough, we used the learned word representations to tag and disambiguate the polysemous words in the text. Finally, we processed the annotated text through the parser pipeline and compared the results with the ones previously obtained with the non-disambiguated corpora.

5.1 Analysis of results

The objective of this thesis was to attest whether it is possible to learn the syntactic structures of a language in an unsupervised manner. Even though the outcomes of our experiments have not yet reached the level of those from supervised methods, the results we obtained were within our expectations given our limitations. Also, the analysis performed over the obtained parses were very helpful in guiding us towards the next steps in this research project, as we will mention in the next section. As a matter of fact, some of these next steps have already begun to be implemented within the team, turning the OpenCog ULL project into a very promising one. In particular, after looking at the evaluation scores obtained from our parser methods and also manually assessing representative examples, there are four main conclusions that stand out:

The first and foremost is that for some corpora we do not have a reliable method for evaluating performances. This is due to the fact that the supervised method we chose to use as our silver standard, the Link Grammar parser in English mode, presented issues in both procedure and performance. In particular, throughout analyzing outputs for the various texts, we discovered that this parser has some tokenization differences with our method which negatively affect the evaluation results. In addition, LG-English returns sometimes non-ideal parses, and comparing against these can significantly reduce the accuracy of our methods. For example, some parses include cycles or some others are counter-intuitive to a human perspective, especially when it comes to complex sentence compositions. Thus, we should not make any confident judgments when using it as a standard. However, in the cases when we did have another standard for comparison, independently of the values it outputted, the relative results of both standards were consistent. Thereupon, we conclude that this supervised parser might be numerically biased as a standard, yet it is useful to get an overall valuation of the performance of our methods against each other.

The second important appraisal we can establish from our results is that the complexity level of the input text is a dominant factor that determines which approach is the best for parsing each particular corpora case. In all of our parsing evaluation tables we see that the sequential parses always obtain better score than random ones, even for the most complex corpora. This shows that language, or more specifically the English language which we explored, has an innate way of structuring itself in a way that pairs of words that are syntactically related tend to be closer to one another; this doesn't come as a surprise given the known success of models such as the n-grams, which only consider a reduced neighboring context in their analysis. On the other hand, we can also see that the more complex the input text is, the lower the results of all our methods get, but also the lower the results for the baselines are. In the case of the sequential baseline, it is simple to understand that more complex text mean less linear structures. In the case of the random parses, the reason lies on the fact that more complex text has longer sentences: this makes the space of random parses grow, making each individual parse less likely. Consequently, for an unsupervised language learning algorithm to be strong, it is important to asses the complexity of the input before processing, and to be able to adapt the strategy of the parser to needs of the corpora.

Moreover, analyzing this for our corpora and methods leads us to the same conclusions. In the case of simple linear texts, the Clique-WIN-dist method seemed to be more reliable because the fixed window it uses forces the links to stay short. However, the bigger and more compound the sentences got, the LG-Any method proved to be better suited for the same reason that the random baseline decreases its effectiveness: with longer sentences, links among more distant words are possible, which are not possible for methods with a fixed window, yet they are still less likely. In this thesis we did not try to vary the window size of the clique method because we did not find the approach as a positive solution to the problem. To our

perspective, wider windows can be adding noise to the count of word-pairs that are not related, while when using LG-Any method, we allow long distance links but much less frequently, thus adding less noise.

The third meaningful conclusion we can deduce from our previous analysis is that accounting for distance between the pairs is essential to the success of an unsupervised parser. In this thesis we tested two ways of accounting for pair distance: in the counting method and in the scoring function. With respect to the first, this statement is supported by the fact that in average, the two OpenCog leading methods were Clique-WIN-Dist and LG-Any; and, as described before, both methods have some way of leading, either implicitly (LG-Any) or explicitly (Clique-WIN-dist), the gathering of word pair statistics towards favoring closer pairs. With respect to the second way, our results corroborate this conclusion when we look at the generally higher scores achieved by FMI-dist against FMI. Here, we also found out that when the sequential baseline is higher, accounting for distance in scoring function makes an even bigger difference. One possible interpretation for this is that accounting for distance in counting is equivalent to accounting for distance in a scoring function that is not linear, but rather in a logarithmic multiplicative way to the FMI of the word pairs. This leaves us with the question of whether testing other scoring functions will give us even better results.

Finally, regarding our efforts in word sense disambiguation, we were able to train embedding models that could disambiguate a selected choice of word senses to some extent: near perfect disambiguation for our proof-of-concept corpus, and less than 80% for Gutenberg Children. Even with those results, disambiguation was able to improve parsing quality, but not by an astounding margin: up to 5% for a given method, in the best cases. This situation suggests the idea that the bottleneck of our process lies within the parsing stage, and that perhaps disambiguation could work more as a fine-tuning mechanism once we have a robust parser (see possible next steps in the next section). This does not mean, however, that using word embedding models is of no use in the pipeline: it remains to be tested if they could be used as a post-parsing step to create vector representations of words that could then be clustered and organized into syntactic categories.

Aside from the previously discussed techniques, one of the greatest results achieved in the course of working with the OpenCog team is that the parsing and disambiguation methods have been integrated as modules into the larger ULL pipeline. Although the results obtained are not comparable to state-of-the-art supervised methods, the possibility to improve each module and see the results is a relevant advantage. The next section reviews some of the possible improvements and paths to follow in the immediate future.

5.2 Recommendations

Based on the analysis of the previous section, we present here possible ways to continue and enhance the syntactic loop of the project in order to advance the general

goal of the project: learning a language grammar in an unsupervised manner.

As noted in section 4.1, a problem while measuring the quality of our automatically generated parses against that of the available standard comes from the different tokenization schemes used by each method. Rule-based parsers like LG include a quantity of language-specific rules for tokenization, while OpenCog uses a simpler method that tries to be language-agnostic. However, for quality-measuring purposes and comparing to the standards more properly, we could try to homogenize tokenization between the MST-parser and the standards used, either LG-English or a treebank. This would give us an idea of how much that factor is affecting our results, but would ultimately mean going back to using the simpler tokenizer, to remain an unsupervised parser.

After initial experiments using FMI, the OpenCog team has started to consider alternative ways of calculating the scoring function of the MST algorithm (see section 3.2). Pair-wise mutual information, as it name implies, considers the relationship between pairs of words that occur within a context. However, it does not deal with any potential higher dependencies that may be relevant in a language syntactic structure. One possible path to follow is to modify our word-pair counting to include higher dependencies. Yet another possibility in the same direction is to use a different scoring function. Given that the search space to look from is too big, we consider a neural-network-generated language model to estimate the weights of links during MST-parsing. In particular, Recurrent Neural Networks (RNN) have been very successful at learning higher order-dependencies [30], so we could use a a language model learned from our corpora by an RNN to estimate the weights for our algorithm.

On a slightly different direction, but still related, another option is to explore a different pair-counting method. From our results we saw that the more complex the corpora, the less linear they were, which is why LG-Any method performed better in those cases. In spite of getting positive results, a possible explanation for it not gathering better information is that not every random tree is equally likely to be a parse. The idea we suggest is to use a treebank to get statistics of what is the probability of a parse with certain types of distance links and then use this as a prior to the counting method to give weights, just as Clique-WIN-dist, but the weights corresponding to the prior and not the distance.

Lastly, in the word sense disambiguation topic, we still have not applied our pipeline to larger corpora like the ones explored in embedding algorithms [2, 27]. Based on the premise that neural networks commonly require a large amount of training data to properly learn their intended function, we could explore simply using our existing processes with a larger corpus, and compare disambiguation and parsing performance against our results. There has also been work on the application of such deep language models that explore longer-term interactions in the field of word sense disambiguation [40]. The OpenCog team is starting to look at options other than AdaGram to perform this, and we will probably continue on these lines.

Bibliography

[1] Leonardo Barazza. How does word2vec's skip-gram work? `https://becominghuman.ai/how-does-word2vecs-skip-gram-work-f92e0525def4`. Medium. Feb 19, 2017.

[2] Sergey Bartunov, Dmitry Kondrashkin, Anton Osokin, and Dmitry Vetrov. Breaking sticks and ambiguities with adaptive skip-gram. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*, pages 130–138, Cadiz, Spain, 09–11 May 2016. PMLR.

[3] Cüneyt F Bazlamaçcı and Khalil S Hindi. Minimum-weight spanning tree algorithms a survey and empirical study. *Computers & Operations Research*, 28(8):767–785, 2001.

[4] Richard Ernest Bellman. *Dynamic programming*. Princeton University Press, 1957.

[5] Yoshua Bengio, Rejean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 3, 2003.

[6] Nan Bernstein-Ratner. The phonology of parent child speech. *Children's language*, 6(3), 1987.

[7] Steven Bird and Edward Loper. Nltk: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 31. Association for Computational Linguistics, 2004.

[8] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[9] Michael R Brent and Timothy A Cartwright. Distributional regularity and phonotactic constraints are useful for segmentation. *Cognition*, 61:93–125, 1996.

[10] Michael R Brent and Jeffrey Mark Siskind. The role of exposure to isolated words in early vocabulary development. *Cognition*, 81(2):B33–B44, 2001.

[11] Jiong Cai, Yong Jiang, and Kewei Tu. Crf autoencoder for unsupervised dependency parsing. *arXiv preprint arXiv:1708.01018*, 2017.

[12] Jose Camacho-Collados and Taher Pilehvar. From word to sense embeddings: A survey on vector representations of meaning. *arXiv preprint arXiv:1805.04032*, 2018.

[13] Xinxiong Chen, Zhiyuan Liu, and Maosong Sun. A unified model for word sense representation and disambiguation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1025–1035, 2014.

[14] Keneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29, 1990.

[15] Shay B Cohen, Kevin Gimpel, and Noah A Smith. Logistic normal priors for unsupervised probabilistic grammar induction. In *Advances in Neural Information Processing Systems*, pages 321–328, 2009.

[16] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[17] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. 1991.

[18] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[19] Emmanuel Dupoux. Cognitive science in the era of artificial intelligence: A roadmap for reverse-engineering the infant language-learner. *Cognition*, 173:43–59, 2018.

[20] Winthrop Nelson Francis. *A manual of information to accompany A standard sample of present-day edited American English, for use with digital computers*. Department of Linguistics, Brown University, 1971.

[21] Ben Goertzel. *Chaotic logic: Language, thought, and reality from the perspective of complex systems science*, volume 9. Springer Science & Business Media, 2013.

[22] Ben Goertzel and Cassio Pennachin. *Artificial general intelligence*, volume 2. Springer, 2007.

[23] Ben Goertzel, Cassio Pennachin, and Nil Geisweiller. *Engineering General Intelligence, Part 1: A Path to Advanced AGI via Embodied Learning and Cognitive Synergy*, volume 5. Springer, 2014.

[24] Zellig S. Harris. Distributional structure. *WORD*, 10(2-3):146–162, 1954.

[25] Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. The goldilocks principle: Reading children's books with explicit memory representations. *arXiv preprint arXiv:1511.02301*, 2015.

[26] Geoffrey E Hinton, James L McClelland, David E Rumelhart, et al. *Distributed representations*. Carnegie-Mellon University Pittsburgh, PA, 1984.

[27] Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 873–882. Association for Computational Linguistics, 2012.

[28] Xuedong Huang, Fileno Alleva, Hsiao-Wuen Hon, Mei-Yuh Hwang, Kai-Fu Lee, and Ronald Rosenfeld. The sphinx-ii speech recognition system: an overview. *Computer Speech & Language*, 7(2):137–148, 1993.

[29] Dan Jurafsky and James H Martin. *Speech and language processing*, volume 3. Pearson London:, 2014.

[30] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. `https://karpathy.github.io/2015/05/21/rnn-efectiveness`. Blog. May 21, 2015.

[31] Dan Klein and Cristopher D. Manning. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *ACL04. Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 479–486. Association for Computational Linguistics, 2004.

[32] Brian MacWhinney. *The CHILDES project: Tools for analyzing talk, Volume II: The database*. Psychology Press, 2014.

[33] Suresh Manandhar and Ioannis P. Klapaftis. Semeval 2010 task 14: Evaluation setting for word sense induction and disambiguation systems. In *DEW '09 Proceedings of the Workshop on Semantic Evaluations: Recent Achievements and Future Directions*, 2009.

[34] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[35] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics, 2005.

[36] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[37] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[38] OpenCog. The open cognition project. `https://wiki.opencog.org/w/The_Open_Cognition_Project`. Accessed: 2017-09-01.

[39] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[40] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[41] Lin Qiu, Yong Cao, Zaiqing Nie, Yong Yu, and Yong Rui. Learning word representation considering proximity and ambiguity. In *AAAI*, pages 1572–1578, 2014.

[42] Steward Rogers. Singularitynet talks collaborative ai as its token sale hits 400% oversubscription.

[43] Kenneth H Rosen. *Discrete mathematics and its applications*, chapter Trees. New York: McGraw-Hill,, 2011.

[44] Jayaram Sethuraman. A constructive definition of dirichlet priors. *Statistica sinica*, pages 639–650, 1994.

[45] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[46] Daniel D.K. Sleator and Davy Temperley. Parsing english with a link grammar. *CMU-CS-91-196*, 1991.

[47] Daniel DK Sleator and Davy Temperley. Parsing english with a link grammar. *arXiv preprint cmp-lg/9508004*, 1995.

[48] Fei Tian, Hanjun Dai, Jiang Bian, Bin Gao, Rui Zhang, Enhong Chen, and Tie-Yan Liu. A probabilistic model for learning multi-prototype word embeddings. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 151–160, 2014.

[49] Linas Vepstas and Ben Goertzel. Learning language from a (large) unnanotated corpus. *ArXiv:1401.3372*, 2014.

[50] Mattias Wahde. *Biologically inspired optimization methods: an introduction*. WIT press, 2008.

[51] Walber. Precision and recall. `https://en.wikipedia.org/wiki/F1_score#/media/File:Precisionrecall.svg`. By Walber [CC BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0)], from Wikimedia Commons. 22 Nov 2014.

[52] Pei Wang and Ben Goertzel. Introduction: Aspects of artificial general intelligence. In *Proceedings of the 2007 conference on Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms: Proceedings of the AGI Workshop 2006*, pages 1–16. IOS Press, 2007.

[53] Arnold D Well and Jerome L Myers. *Research design & statistical analysis*. Psychology Press, 2003.

[54] Roman V. Yampolskiy. Efficient estimation of word representations in vector space. In Xin-She Yang, editor, *Artificial Intelligence, Evolutionary Computation and Metaheuristics (AIECM) –In the footsteps of Alan Turing*, chapter 1, page 3–17. Springer, London, UKPor, 2013.

[55] Deniz Yuret. *Discovery of Linguistic Relations Using Lexical Attraction, PhD thesis*. MIT, 1998.