A Brief Introduction to Natural Language Processing for Non-linguists

Cynthia A. Thompson

CSLI, Ventura Hall,
Stanford University,
Stanford, CA 94305, USA
cthomp@csli.stanford.edu

Abstract. This chapter introduces the field of natural language processing to the computer scientist or logician outside of the field. We first discuss some concepts from the field of linguistics, which focuses on the language half of the NLP equation, then move on to some of the common computational methods used to process and understand language. No previous knowledge of NLP is assumed.

1 Introduction

Natural Language Processing (NLP) is the attempt to use computers to analyze and understand human (as opposed to computer) languages. Applications of this technology include translation, text retrieval, categorization, and summarization, information extraction, and dialogue systems. The field is challenging for many reasons. Computers do not have the tremendous amount of world and contextual knowledge that humans bring to bear when they read a text or participate in a conversation. Also, while we usually do not notice this, language is fraught with ambiguity. Finally, words can be combined into sentences in an infinite variety of ways, making it impossible to simply list the possible uses, contexts, and meanings of a word or phrase (for which new meanings are also constantly being invented).

This chapter introduces the field of natural language processing to the computer scientist or logician outside of the field. NLP is itself a sub-field of the broader field of computational linguistics, which also includes work in language theory. We first discuss some concepts from the field of linguistics, which focuses on the language half of the NLP equation, then move on to some of the common computational methods used to process and understand language. We will only briefly touch on the spoken language recognition and understanding, but focus instead on written language. Finally, the process of generating language with a computer, or *natural language generation*, is a topic that is beyond the scope of this article.

J. Cussens and S. Džeroski (Eds.): LLL'99, LNAI 1925, pp. 36–48, 2000.

2 Linguistic Concepts

The analysis of language is typically divided into four parts: morphology, syntax, semantics, and pragmatics. This section discusses each in turn. A more complete overview of the field can be found in Crystal (1987), and there are many introductory linguistic texts and books of survey papers, for example that by Newmeyer (1988).

Morphology, Phonology, and Words. Morphology is the study of how words can be broken into meaningful pieces, while phonology studies the units of sounds possible in a language, and how they combine with one another. For example, morphological knowledge includes knowledge of suffixes and prefixes. This gives us information such as that the meaning of the word *unbelievable* can be derived from the prefix *un*, the verb *believe*, and the suffix *able*.

We often refer to words as *lexical items*, and a *lexicon* is a data structure used to keep track of information needed to process the words in a sentence. Most theories of language assign *features* to words, such as whether they are singular or plural, transitive or intransitive, or first or third person. These features help detect inconsistencies such as the combination of a singular noun with a plural verb, as in "The dog eat."

Syntax. Syntax is the study of the legal structures of a language. Linguists group words that behave in similar syntactic ways into categories. These categories are called *syntactic categories*, or also *parts of speech* (POS). Categories include noun, verb, and adjective. Knowledge of syntax includes the rules for combining these categories into phrases and the structural roles that these phrases can play in a sentence. Syntax tells us that (1) is an allowable sentence while (2) is not (Linguistics typically use an "*" to indicate a linguistically questionable sentence).

The old man sat on the bench. (1)

∗Man the bench the on sat old. (2)

There are several major phrase types that combine together in various ways to form the phrase structure for the entire sentence. The phrase structure for the sentence "The woman took the book to the park." is displayed in Figure 1. Here we have used the standard POS labels (also called *tags*) from the *Brown corpus,* a free corpus used in much computational linguistics work. The tags used here are sentence (S), noun phrase (NP), verb phrase (VP), past tense verb (VBD), prepositional phrase (PP), and preposition (IN). The most common type of sentence is assigned the category "S," and is divided into a noun phrase and verb phrase.

The rules for combining phrases into larger structures is captured by a *grammar*. The most common type of grammar is a *context-free*, or *phrase structure* grammar. These grammars consist of:

– a set of *terminal symbols* (the lexical items and punctuation),

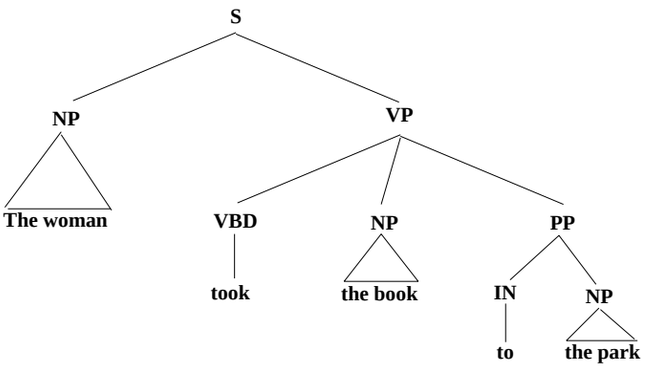

Fig. 1. A Sentence's Phrase Structure

- a set of *non-terminal symbols*, such as the parts of speech,
- a start symbol, and
- a set of *rewrite rules*, each with a non-terminal on the left hand side and zero or more terminal or non-terminals on the right.

A simple example is shown in Table 1.

Table 1. Sample context-free grammar and lexicon

Grammar	**Lexicon**
S → NP VP	Det → a
NP → Det NP2	Det → the
NP → NP2	Noun → dog
NP2 → Noun	Verb → bit
NP2 → NP2 PP	Noun → saw
PP → Prep NP2	Verb → saw
VP → Verb	Prep → with
VP → Verb NP	Noun → woman
VP → VP PP	Verb → eats

Context-free grammars (CFGs) and their variations are widely used, but are not without their problems. First, it is not clear that the syntax of English or any other natural language can be fit into the context-free formalism. Also, handling so called *long-distance dependencies* can complicate the grammar tremendously. An example of such a dependency is in "Whom did Jane take the book from?" where "Whom" serves as the noun phrase in the (implied) prepositional phrase "from whom." This is a dependency because the prepositional phrase that is missing its noun phrase depends on another noun phrase in the sentence, and it is long distance because the intervening structure can be arbitrarily large.

Common alternatives to CFGs include Head-Driven Phrase Structure Grammar, Categorial Grammar, (lexicalized) Tree Adjoining Grammar, and Lexical Functional Grammar. Many of these incorporate features and *unification*. Unification-based grammars take into account the properties (such as number and subcategorization, discussed below) associated with grammatical categories, and attempt to ensure that these properties are consistent where needed across the sentence. Many of these alternative formalisms also incorporate semantics, discussed in the next section.

Another powerful enhancement to context-free grammars is the addition of probabilities to each grammar rule, forming *probabilistic context-free grammars* (PCFGs). PCFGs are useful for several reasons. First, they can derive more than one analysis for an ambiguous sentence, such as "Time flies like an arrow." This sentence, while it has its conventional idiomatic meaning, could be a request to "time flies" as an arrow would do so. While a person might never derive the latter meaning, a straight CFG could. Second, PCFGs allow one to learn grammars from an annotated corpus, a task that would also require examples of non-sentences if one were to try to learn CFGs alone. Such *negative examples* are not very common aside from the starred sentences in linguistic text books! Finally, PCFGs can allow us to parse sentences that ordinarily would not be considered as grammatical, by assigning small probabilities to unlikely but still possible rules.

Besides grammars, an important syntactic notion is that of *dependency*. For example, in the sentence "Joe saw the book in the store." Joe and the book are dependents of a seeing event. They are the *arguments* of the verb see. The prepositional phrase "in the store" is a dependent of book, and modifies book.

When we talk about arguments, we are usually referring to noun phrases as the arguments of verbs. These arguments can be classified by the semantic roles they play. For example, the *agent* of an action is the person or thing that is doing something, and the *patient* is the person or thing that is having something done to it. Another way to classify the arguments is via their syntactic relations such as *subject* and *object*.

Different verbs can take different numbers of arguments, which just means that they may differ in the number of entities that they may describe relationships about. For example, we cannot say "She brought." without giving an object of the bringing.

Verb arguments fall into two categories, the subject (the noun phrase that appears before the verb), and all non-subject arguments, referred to as *complements*. Verbs can be divided into classes depending on what types of complements they allow. This classification is called *subcategorization*. We say that a verb *subcategorizes for* a particular complement. For example, *bring* subcategorizes for an object.

Semantics. Semantics is the study of the meaning of a unit of language, whether it be a word, sentence, or entire discourse. We can divide the study into that of studying the meanings of individual words, and that of how word meanings

combine into sentence meanings (or even larger units). Sentence meanings are studied independently of the broader context in which that sentence is used. Semantics tells us that (3) is an allowable sentence while (4) is not, even though the latter is syntactically correct.

Who took the blue book? (3)

∗The blue book spoke about its small feet. (4)

This kind of distinction is captured by *selectional restrictions*, which describe the semantic regularities associated with the possible complements of a verb. In this example, the verb *speak* prefers people, not animals or books (much less books with feet), as the subject.

Most work in semantics involves a logical representation language. The knowledge of the meaning of a sentence can be equated in this way with knowledge of its *truth conditions,* or what the world would have to be like if the sentence were true.

Pragmatics. Pragmatics is the study of appropriateness, necessity, and sufficiency in language use. It also explores how sentences relate to one another. Pragmatics knowledge includes information about sentences that might have meaning in the present context, but are inappropriate or unnecessary because of the inferences that a rational person could apply.

3 Language Processing

Now we move on to the computational processes that are used at various stages of language processing and understanding. An important distinction in the field is that between those who write programs that are engineered entirely by hand, and which use primarily symbolic techniques, and those who use statistical and machine learning techniques to help engineer their NLP programs. The latter is typically dubbed *statistical NLP*, or *statistical language learning.* There is also a growing movement to combine the two fields, as evidenced by many of the chapters in this volume. For an introduction to natural language understanding in general, Allen (1995) is an excellent reference. Statistical NLP is covered in Charniak (1993) and Manning and Schütze (1999), with a set of recent articles available in Cardie and Mooney (1999).

Before we discuss methods specific to a given level of language processing, several ubiquitous techniques bear mention. One of these is the use of finite state models, including finite state machines, Markov chains, and Hidden Markov Models. These can all be thought of as acceptors or generators for strings of words or other symbols. They each contain states (including a start state), sets of possible events (such as the output of a symbol), and rules for going from one state to the next. A Markov chain models sequences of events where the probability of an event occurring depends upon the fact that a preceding event

occurred. Thus, each transition is associated with the probability of taking that arc, given the current state.

Hidden Markov models (HMMs) have the additional complication that one may not be able to observe the states that the model passes through, but only has indirect evidence about these states. More formally, an HMM is a five-tuple (Π, S, W, T, O), where Π is the start state probability distribution, S is the set of states, W is the set of possible observations, T are transition probabilities between the states, and O are the observation probabilities. HMMs are useful in situations where underlying event probabilities generate surface events. For example, in tagging, discussed next, one can think of underlying chains of possible parts of speech from which actual legal sentences are generated with some probability. There are three questions that are asked about an HMM. First, given a model, how do we (efficiently) compute how likely a certain observation sequence is? Second, given the observation sequence and and a model, how do we choose the state sequence that best explains the observations? Third, given an observation sequence and a space of possible models, how do we find the model that best explains the observed data? Methods for answering these questions, especially the last of learning the probabilities associated with a HMM, have become widespread in the statistical NLP literature. See Rabiner (1989) for a good introduction to HMMs.

Part of Speech Tagging. Part-of-speech (POS) tagging is the process of labeling each word with its part of speech. Tagging is often the first step in the analysis of a sentence, and can assist with later stages of understanding. Three main techniques are used for POS tagging: rule-based tagging, stochastic tagging, and transformation-based tagging. Rule-based techniques use a dictionary or other source to find possible parts of speech, and disambiguation rules to eliminate illegal tag combinations. Stochastic techniques use variants on Hidden Markov Models, based on training from a tagged corpus, to pick the most likely tag for each word. Transformation-based tagging (Brill, 1993) is a technique from machine learning that learns rules for transforming a sentence into its appropriate tags.

Bracketing. Bracketing is the process of dividing up a sentence into its high-level syntactic constituents. For example, for the sentence in Figure 1, we have the partial labeled bracketing:

[S[NP *The woman*]
[VP[VBD *took*][NP *the book*][PP[IN *to*][NP *the library*]]]]

A preliminary bracketing such as this can be useful in guiding a more detailed parse. It can also be used to quickly extract basic information about the structure of a sentence in scenarios where speed of processing is an issue, such as in dialogue systems. Since most systems perform detailed bracketing that is more analogous to a full parse, the process of high-level bracketing will not be further discussed here.

Parsing. Parsing is a more detailed version of bracketing, basically combining the tagging and bracketing steps by labeling the brackets to the lowest level, but the resulting parse is commonly displayed as a tree as in Figure 1. There are many parsing methods in use; we cover only the most common here. A parsing algorithm (*parser*) is a procedure for searching through the possible ways of combining grammatical rules to find one (or more) structure(s) that matches a given sentence's structure. The search for the best parse is constrained both by the grammar and the sentence at hand.

For context free grammars, there are three dimensions along which parsers vary. First, alternative parses can be examined in parallel or sequential fashion. Second, the parser can work in a top-down or bottom-up fashion. Top-down procedures start at the highest structural level (usually sentences), look for rules that make up that structure, and work their way down to the level of the individual words. Bottom-up procedures start with the words and look for rules matching these words, combining them into higher and higher level constituents. Along the third dimension, parsers take different strategies for deciding which pieces of the parse to analyze first. Typical strategies include moving through the input in a set direction, analyzing chunks of increasing size, or combining these methods.

One of the most common algorithms for analyzing sentences using context-free grammars is the *chart parser*. A chart parser contains three structures: a key list (also called the agenda), a chart, and a set of edges. The chart stores the pieces of the parse as it is constructed. Each chart entry contains the name of a terminal or non-terminal symbol in the grammar, the location in the sentence at which that entry begins, and the length of the entry. The key list is a FIFO stack of chart entries that are candidates for entry into the chart. Figure 2 shows a schematic of an empty chart (on the left) and the key list (on the right) for the sentence "The dog eats." The horizontal labels indicate the starting position of a chart entry, and the vertical labels indicate the constituent length. Thus, when "eats, 3, 1" is processed, it is placed in the third column of the first row.

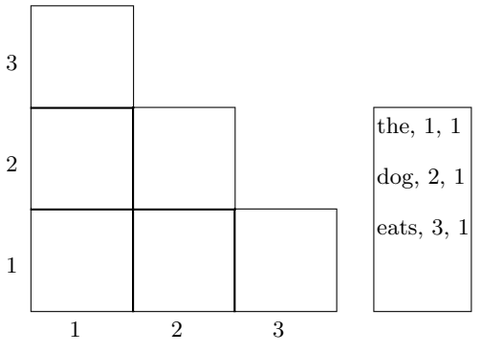

Fig. 2. The chart and key list before parsing begins

The edges keep track of the grammar rules that can be applied to current chart entries to combine them into larger entries. An edge e contains:

- the rule that may be applied ($rule(e)$),
- the sentence position where the first constituent of the right-hand side of the rule was located ($start(e)$),
- the position where the first uncompleted right-hand constituent must start ($end(e)$), and
- an indicator (denoted by $\circ$) in the right-hand side of which constituents have been completed.

An algorithm for chart parsing is given in Table 2.

Table 2. A Chart Parsing Algorithm

Add each word in the sentence to the key list, working from back to front.
While entries in key list do
    Pop a key list entry, c.
    If c is not in the chart, then
        Add c to the chart.
        For all rules that have c's type as the first constituent of their right hand side,
            add an edge, e, for that rule to the chart, where $start(e)$=$start(c)$,
            $end(e)$=$start(c)$+$len(c)$, and $\circ$ is placed after the first constituent.
        For all edges e that have c's type as the constituent following the $\circ$ indicator,
            call `create_extended(`e`, `c`)`
        If the extended edge is completed, add an entry to the key list with
            the appropriate information.

Procedure `create_extended(edge` e`, key list entry` c`)`
    Create a new edge e'.
    Set $start(e') = start(e)$.
    Set $end(e') = start(c)$+$len(c)$.
    Set $rule(e') = rule(e)$ with $\circ$ moved beyond c.

As an example, let us give an overview of the parse of "The dog eats." using the grammar from Table 1. Figure 3 shows the chart (with some empty portions omitted) after processing "the." It was first removed from the key list and added to the chart. Next, an edge was added for all rules that could start with "the." Here we just have the rule "`det` $\rightarrow$ the," which we have added to the bottom of the chart. We thus pictorially indicate $start(e)$ by the tail of the arc and $end(e)$ by the head of the arc. Since "the" is the last constituent of the `det`, the $\circ$ goes at the end of the right-hand side, indicating that it is completed, so we push it onto the key list.

We now start processing the `det` entry. We omit the figure in this case. It would include a new edge in the chart: "`np` $\rightarrow$ `det` $\circ$ `noun`," starting at 1 and

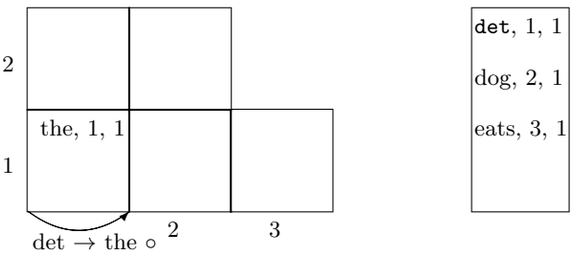

Fig. 3. The chart and key list after processing "the"

ending at 2. It would also include the chart entry "`det`, 1, 1." Next, "dog" is taken from the key list and inserted into the chart, creating a `noun` entry for the key list. In the next step, when we process the noun, we have an example of creating an extended edge. The resulting edge is "`np` → `det noun` ∘," starting at 1 and ending at 3. This finishes the edge, and a new constituent, `np`, is added to the key list. We show the chart after processing `np` in Figure 4. The parse is successful if an `S` constituent is completed that covers the entire sentence.

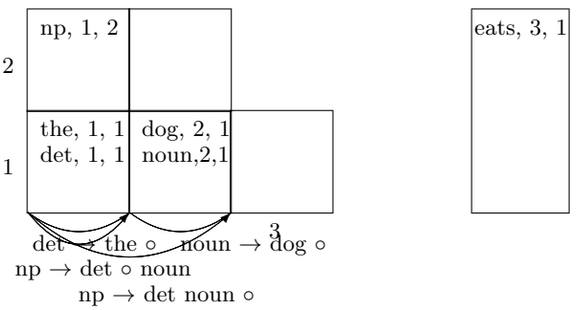

Fig. 4. The chart and key list right before processing "eats"

This is a simplified version of the chart parsers typically used, and does not include a mechanism for actually producing a final syntactic structure for a given sentence. It just indicates success if there exists such a structure. We will not go into the details of maintaining such a structure during the parse, but the process involves some straightforward bookkeeping extensions to the basic chart parser.

A second technique for parsing is the *augmented transition network*. A *transition network* consists of nodes (one of which is the start state) and labeled arcs between the states. An example network for a noun phrase consisting of a determiner, followed by zero or more adjectives, followed by a noun, is shown in Figure 5. Starting in the first node, one can traverse an arc if the current word in the sentence is in the arc's category. Traversing an arc allows us to update

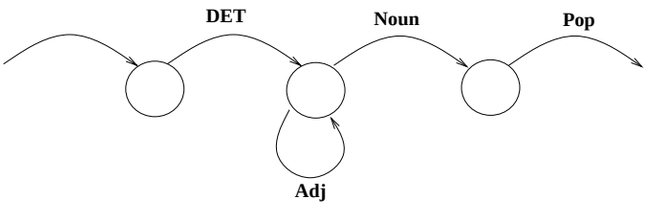

Fig. 5. A Transition Network

the current word, and a phrase is legal if there is a path from the start node to a an arc labeled "Pop". To capture the power of a CFG, one also needs recursion in the grammar, where arcs can refer to other networks, as well as to categories.

A transition network on its own does not handle language phenomena such as agreement and subcategorization. For this we introduce a set of *features* into our grammars, and add conditions and actions to the arcs of a network, creating an augmented transition network. Conditions restrict the circumstances under which an arc can be taken, while actions update the features and structures associated with the parse. Because parsing with augmented transition networks is not as commonly used, we will not discuss the algorithm's details.

Finally, recent work in statistical NLP has tackled the problem of learning PCFGs from a parsed corpus. The first step is to simply enumerate all possible CFG rules by reading them directly from the parsed sentences in the corpus. Next, one attempts to assign some reasonable probabilities to these rules, test them on a new corpus, and remove those with sufficiently small probability. In the latest iteration of such techniques, Charniak (1999) presents a method extracting such probabilities, and the use of the resulting grammar to efficiently parse novel sentences.

Word Sense Disambiguation. The problem addressed by word sense disambiguation algorithms is that many words have several meanings, or *senses*. These words, if taken out of context, thus have several possible interpretations, and are said to be *ambiguous*. A favorite example is "bank," which can refer to the monetary meaning or the river side meaning. The task of word sense disambiguation is to determine which meaning is appropriate in the current context. This task is important for constraining the parsing processes previously discussed, or for assisting with simple word-for-word translations, where the different senses of a word can be translated in different ways. Disambiguation can also be the first stage in semantic processing, discussed below. For an overview of disambiguation methods in general, see Ide and Véronis (1998).

Some of the most successful word sense disambiguation methods to date are from the area of statistical NLP. These can be divided up into supervised learning techniques, dictionary-based methods, and unsupervised techniques. In the supervised learning scenario, we are given a corpus that has the correct sense label for each word, and attempt to learn a model that can correctly label new

sentences (Gale, Church, & Yarowsky, 1992; Brown, Pietra, Pietra, & Mercer, 1991). In dictionary-based techniques, word senses are hypothesized based on an analysis of a knowledge base containing information about words and their relationships, whether it be a dictionary, thesaurus, or parallel corpora in two or more languages (Lesk, 1986; Yarowsky, 1992; Dagan & Itai, 1994). Finally, unsupervised techniques attempt to cluster words into related senses based on the contexts in which they occur or other surface information, e.g., (Schütze, 1995).

Semantic Processing. While all of the above steps are useful, there is still information missing from the syntactic representation and even word senses of a sentence. To allow a system to reason about the implications and meaning of a sentence, it must be transformed into a representation that allows such reasoning. While some systems can use just syntax as a basis for making decisions, deeper interactions require deeper understanding.

Most semantic understanding systems attempt to transform the sentence into an underlying representation language, typically one based on predicate or first order logic. For example "The woman took the book" can be represented as
`(the x: (book x)(the y: (woman y)(took1 y x)))`,
where `the` acts as a quantifier over the variables `x` and `y`. The exact predicates or other representation actually used are highly task-dependent. For example, if a question answering system is the application, the representation might be a database query language. If the task is translation, some sort of interlingua might be the goal. Often an intermediate logical form is used as the result of the semantic interpretation phase, with the final representation being derived from it based on the context.

One of the difficulties of assembling the word meanings into a meaning for the whole sentence is that language does not always obey the principle of *compositionality*. This principle says the meaning of the whole can be strictly predicted from the meaning of the parts. This is clearly not true, as "broad daylight" (light is not wide) and "strong tea" (tea cannot exert a force). These two groupings are examples of *collocations,* or words that tend to appear together or express a conventional way of saying things. Knowing about collocations can help bias the parsing process so that the words in a collocation will not be placed into two different constituents. Collocations are found by counting words in a corpus and noticing which appear together frequently and which could also conceivably be part of the same phrase based on their tags. The details for counting occurrences appropriately are important in getting true collocations, but we will not go into them here.

Some systems simply add semantic features to an existing context-free grammar, and add parsing mechanisms for combining them. This type of technique is common for those taking a theoretical view, but those working on the comprehension of real texts face a problem: real language use includes errors, metaphor, and many other noisy phenomena. As a result, a syntax-driven approach (unless it is probabilistic in nature) may not produce any analysis at all when it cannot find a parse for the entire sentence.

One can overcome these difficulties in many ways. One technique is to first derive a syntactic parse (or partial parse), then use rules to translate this to a semantic representation. Another is to use *semantic grammars*, that translate a sentence directly into its "meaning." These are most useful in limited domains where one can take advantage of the predetermined context to constrain the grammar. Third, in *information extraction* and other constrained tasks, one can specify meaningful patterns that might occur in the text. Information extraction systems attempt to find specific pieces of information in a document, such as the principle actors and objects in a situation.

Deep semantic processing requires the ability to perform inferences based on the representation of the input and the knowledge base used by the system. Existing systems typically work within a narrow domain of understanding to constrain such inferences, such as those of translating parliamentary procedures or helping a user schedule a trip.

4 Summary

We have discussed linguistic concepts and some associated natural language processing techniques. This survey has by no means covered all techniques and applications. In particular, it has almost completely ignored the lowest level processing required for speech recognition, and the highest level processing required for understanding larger units of language such as texts or conversations.

In conclusion, it should be noted that many of the above techniques can and have been used in combination with one another. For example, some methods combine pragmatic knowledge in the semantic parsing process to help constrain the meaning of a sentence based on the context in which it appears. Finally, there are many tasks that are still difficult for current systems, but the state of the art is constantly changing and the future is bright.

References

1. Allen, J. F. (1995). *Natural Language Understanding (2nd Ed.).* Benjamin/Cummings, Menlo Park, CA.
2. Brill, E. (1993). Automatic grammar induction and parsing free text: A transformation-ba sed approach. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pp. 259–265 Columbus, Ohio.
3. Brown, P., Pietra, S. D., Pietra, V. D., & Mercer, R. (1991). Word-sense disambiguation using statistical methods. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pp. 264–270.
4. Cardie, C., & Mooney, R. J. (1999). Machine learning and natural language (introduction to special issue on natural language learning). *Machine Learning, 34*, 5–9.
5. Charniak, E. (1993). *Statistical Language Learning.* MIT Press.

6. Charniak, E. (1999). A maximum-entropy-inspired parser. Tech. rep. CS99-12, Department of Computer Science, Brown University.
7. Crystal, D. (1987). *The Cambridge Encyclopedia of Language*. Cambridge University Press, Cambridge, England.
8. Dagan, I., & Itai, A. (1994). Word sense disambiguation using a second language monolingual corpus. *Computational Linguistics*, *20*, 563–596.
9. Gale, W., Church, K. W., & Yarowsky, D. (1992). A method for disambiguating word senses in a large corpus. *Computers and the Humanities*.
10. Ide, N., & Véronis, J. (1998). Introduction to the special issue on word sense disambiguation: The state of the art. *Computational Linguistics*, *24*(1).
11. Lesk, M. (1986). Automatic sense disambiguation: how to tell a pine cone from an ice cream cone. In *proceedings of the 1986 SIGDOC Conference*, pp. 24–26.
12. Manning, C. D., & Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA.
13. Newmeyer, F. (1988). *Linguistics: The Cambridge Survey*. Cambridge University Press, Cambridge, England.
14. Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257–285.
15. Schütze, H. (1995). Distributional part-of-speech tagging. In *7th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 141–148.
16. Yarowsky, D. (1992). Word-sense disambiguation using statistical methods of Roget's categories trained on large corpora. In *Proceedings of the Fourteenth International Conference on Computational Linguistics*, pp. 454–460.