To Appear in *Computational Linguistics*

Adaptive Multilingual Sentence Boundary Disambiguation

David D. Palmer[*]
The MITRE Corporation

Marti A. Hearst[†]
Xerox PARC

The sentence is a standard textual unit in natural language processing applications. In many languages the punctuation mark that indicates the end-of-sentence boundary is ambiguous; thus the tokenizers of most NLP systems must be equipped with special sentence-boundary recognition rules for every new text collection.

As an alternative, this article presents an efficient, trainable system for sentence boundary disambiguation. The system, called Satz, makes simple estimates of the parts of speech of the tokens immediately preceding and following each punctuation mark, and uses these estimates as input to a machine learning algorithm that then classifies the punctuation mark. Satz is very fast both in training and sentence analysis, and its combined robustness and accuracy surpass existing techniques. The system needs only a small lexicon and training corpus, and has been shown to transfer quickly and easily from English to other languages, as demonstrated on French and German.

1. Introduction

Recent years have seen a dramatic increase in the availability of online text collections. One active area of research is the development of algorithms for aligning sentences in parallel corpora. The success of most Natural Language Processing (NLP) algorithms, including multilingual sentence alignment algorithms (Kay and Röscheisen, 1993; Gale and Church, 1993)[1], part-of-speech taggers (Cutting et al., 1991), and parsers, depends on prior knowledge of the location of sentence boundaries.

Segmenting a text into sentences is a non-trivial task, however, since in English and many other languages the end-of-sentence punctuation marks[2] are ambiguous. A period, for example, can denote a decimal point, an abbreviation, the end of a sentence, or even an abbreviation at the end of a sentence. Exclamation points and question marks can occur within quotation marks or parentheses as well as at the end of a sentence. Ellipsis (a series of periods (...)), can occur both within sentences and at sentence boundaries. The ambiguity of these punctuation marks is illustrated in the following difficult cases:

(1) The group included Dr. J.M. Freeman and T. Boone Pickens Jr.

[*] 202 Burlington Road, Bedford, MA 01730, E-mail: palmer@mitre.org. Some of the work reported here was done while the author was at the University of California, Berkeley. The views and opinions in this paper are those of the authors and do not reflect the MITRE Corporation's current work position.

[†] 3333 Coyote Hill Rd., Palo Alto, CA 94304, E-mail: hearst@parc.xerox.com

[1] There is some recent research in aligning bilingual corpora without relying on sentence boundaries (Fung and Church, 1994; Fung and McKeown, 1994).

[2] In this article, we will consider only the period, the exclamation point and the question mark to be possible "end-of-sentence punctuation marks," and all references to "punctuation marks" will refer to these three. Although the colon, the semicolon, and conceivably the comma can also delimit grammatical sentences, their usage is beyond the scope of this work.

(2) "This issue crosses party lines and crosses philosophical lines!" said Rep. John Rowland (R., Conn.).

(3) Somit entsprach ein ECU am 17. 9. 1984 0.73016 US$ (vgl. Tab. 1).

(4) Créé au début des années 60 ... ... par un gouvernement conservateur : ... cet Office s'était vu accorder six ans ...

The existence of punctuation in grammatical subsentences suggests the possibility of a further decomposition of the sentence boundary problem into types of sentence boundaries, one of which would be "embedded sentence boundary." Such a distinction might be useful for certain applications which analyze the grammatical structure of the sentence. However, in this work we will only address the simpler problem of determining boundaries *between* sentences, finding that which Nunberg (1990) calls the "text-sentence".

In examples (1-4), the word immediately preceding and the word immediately following a punctuation mark provide important information about its role in the sentence. However, more context may be necessary, such as when punctuation occurs in a subsentence within quotation marks or parentheses, as seen in example (2), or when an abbreviation appears at the end of a sentence, as seen in (5a):

(5a) It was due Friday by 5 p.m. Saturday would be too late.

(5b) She has an appointment at 5 p.m. Saturday to get her car fixed.

Examples (5a-b) also show some problems inherent in relying on brittle features, such as capitalization, when determining sentence boundaries. The initial capital in *Saturday* does not necessarily indicate that *Saturday* is the first word in the sentence. As a more dramatic example, some important kinds of text consist only of upper-case letters, thus thwarting any system that relies on capitalization rules. Another obstacle to systems that rely on brittle features is that many texts are not well-formed. One such class of texts are those that are found as the output of optical character recognition (OCR); typically these texts contain many extraneous or incorrect characters.

This article presents an efficient, trainable system for sentence boundary disambiguation that circumvents these obstacles. The system, called Satz, makes simple estimates of the parts of speech of the tokens immediately preceding and following each punctuation mark, and uses these estimates as input to a machine learning algorithm that determines whether the punctuation mark is a sentence boundary or serves another purpose in the sentence. Satz is very fast both in training and sentence analysis; training is accomplished in less than one minute on a workstation, and it can process 10,000 sentences per minute. The combined robustness and accuracy of the system surpass existing techniques, consistently producing an error rate less than 1.5% on a range of corpora and languages. It requires only a small lexicon (which can be less than 5,000 words) and a training corpus of 300-500 sentences.

The following sections discuss related work and the criteria used to evaluate such work, describe the system in detail, and present the results of applying the system to a variety of texts. The transferability of the system from English to other languages is also demonstrated on French and German text. Finally, the learning-based system is shown to be able to improve the results of a more conventional system on especially difficult cases.

2. Related Work

2.1 Evaluation of related work

An important consideration when discussing related work is the mode of evaluation. To aid our evaluation, we define a *lower bound*, an objective score which any reasonable algorithm should be able to match or better. In our test collections, the ambiguous punctuation mark is used much more often as a sentence boundary marker than for any other purpose. Therefore, a very simple, successful algorithm is one in which every potential boundary marker is labeled as the end-of-sentence. Thus, for the task of sentence boundary disambiguation, we define the lower bound of a text collection as the percentage of possible sentence-ending punctuation marks in the text which indeed denote sentence boundaries.

Since the use of abbreviations in a text depends on the particular text and text genre, the number of ambiguous punctuation marks, and the corresponding lower bound, will vary dramatically depending on text genre. For example, Liberman and Church (1992) report on a Wall Street Journal corpus containing 14,153 periods per million tokens, whereas in the Tagged Brown corpus (Francis and Kucera, 1982), the figure is only 10,910 periods per million tokens. Liberman and Church also report that 47% of the periods in the WSJ corpus denote abbreviations (thus a lower bound of 53%), compared to only 10% in the Brown corpus (lower bound 90%)(Riley, 1989). In contrast, Müller(1980) reports lower bound statistics ranging from 54.7% to 92.8% within the a corpus of scientific abstracts. Such a range of lower bound figures suggests the need for a robust approach that can adapt rapidly to different text requirements.

Another useful evaluation technique is the comparison of a new algorithm against a strong *baseline algorithm.* The baseline algorithm should perform better than the lower bound and should represent a strong effort or a standard method for solving the problem at hand.

Although sentence boundary disambiguation is an essential preprocessing step of many natural language processing systems, it is a topic rarely addressed in the literature and there are few public-domain programs for performing the segmentation task. For our studies we compared our system against the results of the UNIX STYLE program (Cherry and Vesterman, 1991).[3] The STYLE program, which attempts to provide a stylistic profile of writing at the word and sentence level, reports the length and structure for all sentences in a document, thereby indicating the sentence boundaries. STYLE defines a sentence as a string of words ending in one of: period, exclamation point, question mark, or backslash-period (the latter of which can be used by an author to mark an imperative sentence ending). The program handles numbers with embedded decimal points and commas and makes use of an abbreviation list with 48 entries. It also uses the following heuristic: initials cause a sentence break only if the next word begins with a capital letter and is found in a dictionary of function words. In an evaluation on a sample of 20 documents, the developers of the program found it to incorrectly classify sentence boundaries 204 times out of 3287 possible (an error rate of 6.3%).

2.2 Regular expressions and heuristic rules

The method currently widely used for determining sentence boundaries is a regular grammar, usually with limited lookahead. In the simplest implementation of this method, the grammar rules attempt to find patterns of characters, such as "period-space-capital letter" which usually occur at the end of a sentence. More elaborate implementations, such

[3] Comparison against the STYLE program was suggested to us by Mickey Chandrasekar.

as the STYLE program discussed above, consider the entire word preceding and following the punctuation mark and include extensive word lists and exception lists to attempt to recognize abbreviations and proper nouns. There are a few examples of rule-based and heuristic systems for which performance numbers are available, discussed in the remainder of this subsection.

The Alembic information extraction system (Aberdeen et al., 1995) contains a very extensive regular-expression-based sentence boundary disambiguation module, created using the lexical scanner generator Flex (Nicol, 1993). The boundary disambiguation module is part of a comprehensive preprocess pipeline which utilizes a list of 75 abbreviations and a series of over 100 hand-crafted rules to identify sentence boundaries, as well as titles, date and time expressions, and abbreviations. The sentence boundary module was developed over the course of more than 6 staff months. On the Wall Street Journal corpus described in Section 4, Alembic achieved an error rate of 0.9%.

Christiane Hoffmann (1994) used a regular expression approach to classify punctuation marks in a corpus of the German newspaper *die tageszeitung* with a lower bound (as defined above) of 92%. She used the UNIX *lex* program (Lesk and Schmidt, 1975) and a large abbreviation list to classify occurrences of periods. Her method incorrectly classified less than 2% of the sentence boundaries when tested on 2827 periods from the corpus. The method was developed specifically for the *tageszeitung* corpus, and Hoffmann reports that success in applying her method to other corpora would be dependent on the quality of the available abbreviation lists. Her work would therefore probably not be easily transportable to other corpora or languages.

Mark Wasson and colleagues invested 9 staff months developing a system that recognizes special tokens (e.g., non-dictionary terms such as proper names, legal statute citations, etc.) as well as sentence boundaries. From this, Wasson built a stand-alone boundary recognizer in the form of a grammar converted into finite automata with 1419 states and 18002 transitions (excluding the lexicon). The resulting system, when tested on 20 megabytes of news and case law text achieved an error rate of 0.3% at speeds of 80,000 characters per CPU second on a mainframe computer. When tested against upper-case legal text the system still performed very well, achieving error rates of 0.3% and 1.8% on test data of 5305 and 9396 punctuation marks, respectively. According to Wasson, it is not likely, however, that the results would be this strong on lower-case-only data.[4]

Although the regular grammar approach can be successful, it requires a large manual effort to compile the individual rules used to recognize the sentence boundaries. Such efforts are usually developed specifically for a text corpus (Liberman and Church, 1992; Hoffmann, 1994) and would probably not be portable to other text genres. Because of their reliance on special language-specific word lists, they are also not portable to other natural languages without repeating the effort of compiling extensive lists and rewriting rules. In addition, heuristic approaches depend on having a well-behaved corpus with regular punctuation and few extraneous characters, and they would probably not be very successful with texts obtained via optical character recognition (OCR).

Müller (1980) provides an exhaustive analysis of sentence boundary disambiguation as it relates to lexical endings and the identification of abbreviations and words surrounding a punctuation mark, focusing on text written in English. This approach makes multiple passes through the data to find recognizable suffixes and thereby filters out

[4] This work has not been published. All information about this system is courtesy of a personal communication with Mark Wasson. Wasson's reported processing time cannot be compared directly to the other systems since it was obtained from a mainframe computer and was estimated in terms of characters rather than sentences.

words which aren't likely to be abbreviations. The morphological analysis makes it possible to identify words which are not otherwise present in the extensive word lists used to identify abbreviations. Error rates of 2-5% are reported for this method tested on over 75,000 scientific abstracts, with lower bounds ranging from 54.7% to 92.8%.

2.3 Approaches using machine learning

There have been two other published attempts to apply machine-learning techniques to the sentence boundary disambiguation task. Both make use of the words in the context found around the punctuation mark.

2.3.1 Regression trees. Riley (1989) describes an approach that uses regression trees (Breiman et al., 1984) to classify periods according to the following features:

- Probability[word preceding "." occurs at end of sentence]
- Probability[word following "." occurs at beginning of sentence]
- Length of word preceding "."
- Length of word after "."
- Case of word preceding ".": Upper, Lower, Cap, Numbers
- Case of word following ".": Upper, Lower, Cap, Numbers
- Punctuation after "." (if any)
- Abbreviation class of words with "."

The method uses information about one word of context on either side of the punctuation mark and thus must record, for every word in the lexicon, the probability that it occurs next to a sentence boundary. Probabilities were compiled from 25 million words of pre-labeled training data from a corpus of AP newswire. The probabilities were actually estimated for the beginning and end of *paragraphs* rather than for all sentences, since paragraph boundaries were explicitly marked in the AP corpus, while the sentence boundaries were not. The resulting classification tree was used to identify whether a word ending in a period is at the end of a declarative sentence in the Brown corpus, and achieved an error rate of 0.2%.[5] Although this is an impressive error rate, the amount of training data and the memory required to store and access probabilities on 25 million word tokens makes this method overly cumbersome for a problem that acts as a preprocessing step to other natural language processing tasks, and would also require a large set of statistics for every corpus and language to be tagged.

2.3.2 Feed-forward neural networks. Humphrey and Zhou (1989) report using a feed-forward neural network to disambiguate periods, and achieve an error rate averaging 7%. They use a regular grammar to tokenize the text before training the neural nets, but no further details of their approach are available.[6]

[5] Time for training was not reported, nor was the amount of the Brown corpus against which testing was performed; we assume the entire Brown corpus was used. Furthermore, no estimates of scalability were given, so we are unable to report results with a smaller set.

[6] Results were obtained courtesy of a personal communication with Joe Zhou.

2.4 Our approach

Each of the approaches described above has disadvantages to overcome. In the following sections we present an approach that avoids the problems of previous approaches, yielding a very low error rate and behaving more robustly than solutions that require manually designed rules. We present results of testing our system on several corpora in three languages: English, German, and French.

3. The Satz System

This section describes the structure of our adaptive sentence boundary disambiguation system, known as Satz.[7] The Satz system represents the context surrounding a punctuation mark as a sequence of vectors, where the vector constructed for each context word represents an estimate of the *part-of-speech distribution* for the word, obtained from a lexicon containing part-of-speech frequency data. This use of part-of-speech estimates of the context words, rather than the words themselves, is a unique aspect of the Satz system, and is responsible in large part for its efficiency and effectiveness.

The context vectors, or "descriptor arrays," are input to a machine learning algorithm trained to disambiguate sentence boundaries. The output of the learning algorithm is then used to determine the role of the punctuation mark in the sentence. The architecture of the system is shown in Figure 1. The Satz system works in two modes – learning mode and disambiguation mode. In learning mode, the input text is a training text with all sentence boundaries manually labeled, and the parameters in the learning algorithm are dynamically adjusted during training. Once learning mode is completed, the parameters in the learning algorithm remain fixed. Training of the learning algorithm is therefore necessary only once for each language, although training can be repeated for a new corpus or genre within a language, if desired. In disambiguation mode, the input is the text whose sentence boundaries have not been marked up yet and need to be disambiguated.

The essence of the Satz system lies in how machine learning is used, rather than in which particular method is used. In this article we report results using two different learning methods: neural networks and decision trees. The two methods are almost equally effective for this task, and both train and run quickly using small resources. However, for some applications one may be more appropriate than another, (e.g., the scores produced by a neural net may be useful for another processing step in a natural language program), so we do not consider either learning algorithm to be the "correct" one to use. Therefore, when we refer to the Satz system, we refer to the use of machine learning with a small training corpus, representing the word context surrounding each punctuation mark in terms of estimates of the parts of speech of those words, where these estimates are derived from a very small lexicon.

3.1 Tokenization

The first stage of the process is lexical analysis, which breaks the input text (a stream of characters) into tokens. The Satz tokenizer is implemented using the UNIX tool LEX (Lesk and Schmidt, 1975) and is modeled on the tokenizer used by the PARTS part-of-speech tagger (Church, 1988). The tokens returned by the lex program can be a sequence of alphabetic characters, a sequence of digits,[8] or a sequence of one or more non-alphanumeric characters such as a periods or quotation marks.

[7] "Satz" is the German word for "sentence."

[8] Numbers containing periods acting as decimal points are considered a single token. This eliminates one possible ambiguity of the period at the lexical analysis stage.

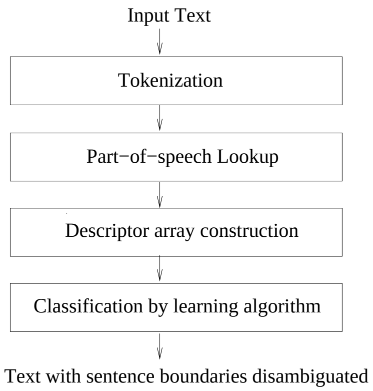

Input Text
Tokenization
Part−of−speech Lookup
Descriptor array construction
Classification by learning algorithm
Text with sentence boundaries disambiguated

Figure 1
The Satz Architecture.

3.2 Part-of-speech lookup

The individual tokens are next assigned a series of possible parts of speech, based on a lexicon and simple heuristics described below.

3.2.1 Representing context. The context surrounding a punctuation mark can be represented in various ways. The most straightforward is to use the individual words preceding and following the punctuation mark, as in this example:

at the plant. He had thought

Using this approach, a representation of an individual word's position in a context must be made for every word in the language. Compiling these representations for each word is undesirable due to the large amount of training data, training time, and storage overhead required, especially since it is unlikely that such information will be useful to later stages of processing.

As an alternative, the context could be approximated by using a single part-of-speech for each word. The above context would then be represented by the following part-of-speech sequence:

preposition article noun

pronoun verb verb

However, requiring a single part-of-speech assignment for each word introduces a processing circularity: because most part-of-speech taggers require predetermined sentence boundaries, the boundary disambiguation must be done before tagging. But if the disambiguation is done before tagging, no part-of-speech assignments are available for the boundary-determination system. To avoid this circularity, we approximate each word's part of speech in one of two ways: (1) by *the prior probabilities* of all parts of speech for that word, or (2) by a *binary value* for each possible part of speech for that word.

In the case of prior probabilities, each word in the context is represented by the probability that the word occurs as each part-of-speech, with all part-of-speech probabilities in the vector summing to 1.0. Continuing the example, the context becomes (and for simplicity, suppressing the parts of speech with value 0.0):

preposition(1.0) *article(1.0)* *noun(0.8)/verb(0.2)*

pronoun(1.0) *verb(1.0)* *noun(0.1)/verb(0.9)*

This denotes that “at” and “the” have a probability of 1.0 of occurring as a preposition and article respectively, “plant” has a probability of 0.8 of occurring as a noun and a probability of 0.2 of occurring as a verb, and so on. These “probabilities,” which are more accurately “scaled frequencies,” are based on occurrences of the words in a pre-tagged corpus[9], and are therefore corpus dependent.

In the case of binary part-of-speech assignment, for each possible part of speech, the vector is assigned the value 1 if the word can ever occur as that part-of-speech (according to the lexicon), and the value 0 if it cannot. In this case the sum of all items in the vector is not predefined, as is the case with probabilities. Continuing the example with binary POS vectors (and, for simplicity, suppressing the parts of speech with value 0), the context becomes:

preposition(1)   article(1)   noun(1)/verb(1)

pronoun(1)   verb(1)   noun(1)/verb(1)

The part-of-speech data necessary to construct probabilistic and binary vectors is often present in the lexicon of a part-of-speech tagger or other existing NLP tool, or it can easily be obtained from word lists; the data would thus be readily available and would not require excessive storage overhead. It is also possible to estimate part-of-speech data for new or unknown words. For these reasons we chose to approximate the context in our system by using the prior part-of-speech information. In Section 4.7 we give the results of a comparative study of system performance with both probabilistic and binary part-of-speech vectors.

3.2.2 The lexicon. An important component of the Satz system is the lexicon containing part-of-speech frequency data from which the descriptor arrays are constructed. Words in the lexicon are followed by a series of part-of-speech tags and associated frequencies, representing the possible parts of speech for that word and the frequency with which the word occurs as each part of speech. The frequency information can be obtained in various ways, as discussed in the previous section. The lexical lookup stage of the Satz system finds a word in the lexicon (if it is present) and returns the possible parts of speech. For the English word *well*, for example, the lookup module might return the tags

JJ/15 NN/18 QL/68 RB/634 UH/22 VB/5

indicating that, in the corpus on which the lexicon is based,[10] the word *well* occurred 15 times as an adjective, 18 as a singular noun, 68 as a qualifier, 634 as an adverb, 22 as an interjection, and 5 as a singular verb.

3.2.3 Heuristics for unknown words. If a word is not present in the lexicon, the Satz system contains a set of heuristics which attempt to assign the most reasonable parts of speech to the word. A summary of these heuristics is listed below.

- Unknown tokens containing a digit (0-9) are assumed to be numbers.
- Any token beginning with a period, exclamation point, or question mark is assigned a “possible end-of-sentence punctuation” tag. This catches common sequences like “?!” and “...”

[9] The frequencies can be obtained from an existing corpus tagged manually or automatically; the corpus does not need to be tagged specifically for this task.

[10] In this example, the frequencies are derived from the Brown corpus (Francis and Kucera, 1982).

- Common morphological endings are recognized and the appropriate part(s)-of-speech is assigned to the entire word.

- Words containing a hyphen are assigned a series of tags and frequencies equally distributed between adjective, common noun, and proper noun.

- Words containing an internal period are assumed to be abbreviations.

- A capitalized word is not always a proper noun, even when it appears somewhere other than in a sentence's initial position (e.g., the word "American" is often used as an adjective). Those words not present in the lexicon are assigned a certain language-dependent probability (0.9 for English) of being a proper noun, and the remainder is distributed uniformly among adjective, common noun, verb, and abbreviation, the most likely tags for unknown words.[11]

- Capitalized words appearing in the lexicon but not registered as proper nouns can nevertheless still be proper nouns. In addition to the part-of-speech frequencies present in the lexicon, these words are assigned a certain probability of being a proper noun (0.5 for English) with the probabilities already assigned to that word redistributed proportionally in the remaining 0.5. The proportion of words falling into this category varies greatly depending on the style of the text and the uniformity of capitalization.

- As a last resort, the word is assigned the set of all possible tags with a uniform frequency distribution.

These heuristics can be easily modified and adapted to the specific needs of a new language,[12] although we obtained low error rates without changing the heuristics.

3.3 Descriptor array construction

A vector is constructed for each token in the input text. This vector is known as a *descriptor array.* The lexicon may contain as many as several hundred very specific tags, which we first need to map into more general categories. For example, the Brown corpus tags of *present tense verb*, *past participle*, and *modal verb* are all mapped into the more general "verb" category. The parts of speech returned by the lookup module are thus mapped into the 18 general categories given in Figure 2, and the frequencies for each category are summed. In the case of a probabilistic vector described in Section 3.2.1, the 18 category frequencies for the word are then converted to probabilities by dividing the frequencies for each by the total frequency for the word. For a binary vector, all categories with a non-zero frequency count are assigned a value of 1, and all others are assigned a value of 0. In addition to the 18 category frequencies, the descriptor array also contains two additional flags that indicate if the word begins with a capital letter and if it follows a punctuation mark, for a total of 20 items in each descriptor array. These last two flags allow the system to include capitalization information when it is available without having to require that this information be present.

[11] Note that in the case of binary vectors, all "probabilities" receive the value 1.

[12] For example, the probability of a capitalized word being a proper noun is higher in English than in German, where all nouns are also capitalized.

noun
article
conjunction
preposition
number
left parentheses
non-punctuation character
colon or dash
sentence-ending punctuation
verb
modifier
pronoun
proper noun
comma or semicolon
right parentheses
possessive
abbreviation
others

Figure 2
Elements of the descriptor array assigned to each incoming token.

3.4 Classification by a learning algorithm

The descriptor arrays representing the tokens in the context are used as the input to a machine learning algorithm. To disambiguate a punctuation mark given a context of k surrounding words (referred to in this article as “k-context”), a window of $k + 1$ tokens and their descriptor arrays is maintained as the input text is read. The first $k/2$ and final $k/2$ tokens of this sequence represent the context in which the middle token appears. If the middle token is a potential end-of-sentence punctuation mark, the descriptor arrays for the context tokens are input to the learning algorithm and the output result indicates whether the punctuation mark serves as a sentence boundary or not. In learning mode, the descriptor arrays are used to train the parameters of the learning algorithm. We investigated the effectiveness of two separate algorithms: (1) back-propagation training of neural networks, and (2) decision tree induction. The learning algorithms are described in the next two sections, and the results obtained with the algorithms are presented in Section 4.

3.4.1 Neural network. Artificial neural networks have been successfully applied for many years in speech recognition applications (Bourland and Morgan, 1994; Lippmann, 1989), and more recently in NLP tasks such as word category prediction (Nakamura et al., 1990) and part-of-speech tagging (Schmid, 1994). Neural networks in the context of machine learning provide a well-tested training algorithm (back-propagation) that has achieved high success rates in pattern-recognition problems similar to the problem posed by sentence boundary disambiguation (Hertz, Krogh, and Palmer, 1991).

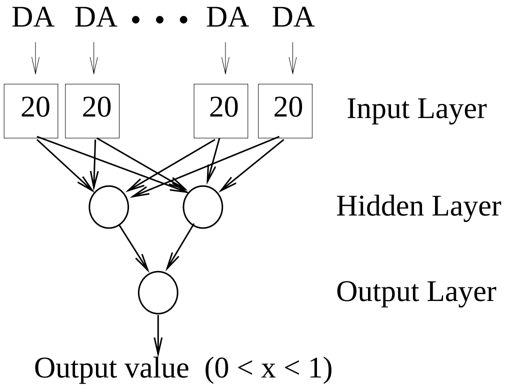


Figure 3
Neural Network Architecture (DA = descriptor array of 20 items)

For Satz, we used a fully-connected feed-forward neural network, as shown in Figure

3. The network accepts $k * 20$ input values, where k is the size of the context and 20 is the number of elements in the descriptor array described in Section 3.3. The input layer is fully connected to a hidden layer consisting of j hidden units; the hidden units in turn feed into one output unit which indicates the results of the function. In a traditional back-propagation network, the input to a node is the sum of the outputs of the nodes in the previous layer multiplied by the weights between the layers. This sum is then passed through a “squashing” function to produce a node output between 0 and 1. A commonly-used squashing function, due to its mathematical properties which assist in network training, is the sigmoidal function, given by $f(h_i) = \frac{1}{1+e^{-h_i/T}}$, where h_i is the node input and T is a constant to adjust the slope of the sigmoid.

In the Satz system we use a sigmoidal squashing function on all hidden nodes and the single output node of the neural network. The output of the network is thus a single value between 0 and 1, and represents the strength of the evidence that a punctuation mark occurring in its context is indeed the end of a sentence. Two adjustable sensitivity thresholds, t_0 and t_1, are used to classify the results of the disambiguation. If the output is less than t_0, the punctuation mark is not a sentence boundary; if the output is greater than or equal to t_1, it is a sentence boundary. Outputs which fall between the thresholds cannot be disambiguated by the network (which may indicate that the mark is inherently ambiguous) and are marked accordingly, so they can be treated specially in later processing.[13] For example, the sentence alignment algorithm in (Gale and Church, 1993) allows a distinction between *hard* and *soft* boundaries, where soft boundaries are movable by the alignment program. In our case, punctuation marks remaining ambiguous after processing by Satz can be treated as soft boundaries while unambiguous punctuation marks (as well as paragraph boundaries) can be treated as hard boundaries, thus allowing the alignment program greater flexibility.

A neural network is trained by presenting it with input data paired with the desired output. For Satz, the input is the context surrounding the punctuation mark to be disambiguated, and the output is a score indicating how much evidence there is that the punctuation mark is acting as an end of sentence boundary. The nodes are connected via links that have weights assigned to them, and if the network produces an incorrect score, the weights are adjusted using an algorithm called *back-propagation* (Hertz, Krogh, and Palmer, 1991) so that the next time the same input is presented to the network, the output should more closely match the desired score. This training procedure is often iterated many times in order to allow the weights to adjust appropriately, and the same input data is presented multiple times. Each round of presenting the same input data is called an *epoch*; of course, it is desirable to require as few training epochs and as little training data as possible. If one trains the network too often on the same data, *overfitting* can occur, meaning that the weights become too closely aligned with the particular training data that has been presented to the network, and so may not correspond well to new examples that will come later. For this reason, training should be accompanied by what's known as *cross-validation* (Bourland and Morgan, 1994), a check against a held-out set of data to be sure that the weights are not too closely tailored to the training text. This will be described in more detail below.

Training data for the neural network consist of two sets of text in which all sentence boundaries have been manually disambiguated. The first text, the *training text*, contains 300-600 *test cases*, where a test case is an ambiguous punctuation mark. The weights of the neural network are trained on the training text using the standard back-propagation algorithm (Hertz, Krogh, and Palmer, 1991). The second set of texts used in training

[13] When $t_0 = t_1$, no punctuation mark is left ambiguous.

is the cross-validation set whose contents are separate from the training text and which contains roughly half as many test cases as the training text. Training of the weights is not performed on this text; the cross-validation text is instead used to increase the generalization of the training, such that when the total training error[14] over the cross-validation text reaches a minimum, training is halted. Testing is then performed on texts independent of the training and cross-validation texts. We measure the speed of training by the number of training epochs required to complete training, where an epoch is a single pass through all the training data. Training times for all experiments reported in this article were less than one minute and were obtained on a DEC Alpha 3000 workstation, unless otherwise noted.

In Sections 4.1-4.9 we present results of testing the Satz system with a neural network, including investigations of the effects of varying network parameters such as hidden layer size, threshold values, and amount of training data.

3.4.2 Decision tree. Algorithms for decision tree induction (Quinlan, 1986; Bahl et al., 1989) have been successfully applied to NLP problems such as parsing (Resnik, 1993; Magerman, 1995) and discourse analysis (Siegel and McKeown, 1994; Soderland and Lehnert, 1994). We tested the Satz system using the c4.5 (Quinlan, 1993) decision tree induction program as the learning algorithm and compared the results to those obtained previously with the neural network. These results are discussed in Section 4.10.

The induction algorithm proceeds by evaluating the information content of a series of binary *attributes* and iteratively building a tree from the attribute values, with the leaves of the decision tree being the values of the *goal attributes*. At each step in the learning procedure, the evolving tree is branched on the attribute which divides the data items with the highest gain in information. Branches are added to the tree until the decision tree can classify all items in the training set. Overfitting is also possible in decision tree induction, resulting in a tree that can very accurately classify the training data but may not be able to accurately classify new examples. To reduce the effects of overfitting, the c4.5 learning algorithm *prunes* the tree after the entire decision tree has been constructed. It recursively examines each subtree to determine whether replacing it with a leaf or a branch would reduce the number of errors. This pruning produces a decision tree which is better able to classify data different from the training data.

Integrating the decision tree induction algorithm into the Satz system was simply a matter of defining the input attributes as the k descriptor arrays in the context, with a single goal attribute representing whether the punctuation mark is a sentence boundary or not. Training data for the induction of the decision tree were identical to the training set used to train the neural network.

4. Experiments with English Texts

We first tested the Satz system using English texts from the Wall Street Journal portion of the ACL/DCI collection (Church and Liberman, 1991). We constructed a training text of 573 test cases and a cross-validation text of 258 test cases.[15] We then constructed a separate test text consisting of 27,294 test cases, with a lower bound of 75.0%. The

[14] The training error is the least mean squares error, one-half the sum of the squares of all the errors, where the error of a particular item is the difference between the desired output and the actual output of the neural net.

[15] Note that "constructing" a training, cross-validation, or test text simply involves manually disambiguating the sentence boundaries by inserting a unique character sequence at the end of each sentence.

baseline system (UNIX STYLE) achieved an error rate of 8.3% on the sentence boundaries in the test set. The lexicon and thus the frequency counts used to calculate the descriptor arrays were derived from the Brown corpus (Francis and Kucera, 1982). In initial experiments we used the extensive lexicon from the PARTS part-of-speech tagger (Church, 1988) which contains 30,000 words. We later experimented with a much smaller lexicon, and these results are discussed in Section 4.4. In Sections 4.1 - 4.9 we describe the results of our experiments with the Satz system using the neural network as the learning algorithm. Section 4.10 describes results using decision tree induction.

4.1 Context size

In order to determine how much context is necessary to accurately disambiguate sentence boundaries in a text, we varied the size of the context from which the neural network inputs were constructed and obtained the results in Table 1. The number in the *Training Epochs* column is the number of passes through the training data required to learn the training set; the number in the *Testing Errors* column is the number of errors on the 27,294 item test set the system made after training with the corresponding context size. From these data we concluded that a 6-token context, 3 preceding the punctuation mark and 3 following, produces the best results.

Table 1
Results of comparing context sizes.

Context Size	Training Epochs	Testing Errors	Error (%)
4-context	1731	1424	5.2%
6-context	218	409	1.5%
8-context	831	877	3.2%

4.2 Hidden units

The number of hidden units in a neural network can affect its performance. To determine the size of the hidden layer in the neural network which produced the lowest output error rate, we experimented with various hidden layer sizes and obtained the results in Table 2. From these data we concluded that the lowest error rate in this case is possible using a neural network with two nodes in its hidden layer.

Table 2
Results of comparing hidden layer sizes (6-context).

# Hidden Units	Training Epochs	Testing Errors	Error (%)
1	623	721	2.6%
2	216	409	1.5%
3	239	435	1.6%
4	350	1343	4.9%

4.3 Sources of errors

As described in Sections 4.1 and 4.2, the best results were obtained with a context size of 6 tokens and a hidden layer with 2 units. This configuration produced a total of 409 errors out of 27,294 test cases, for an error rate of 1.5%. These errors fall into two major categories: (i) "false positive", i.e., a punctuation mark the method erroneously labeled as a sentence boundary, and (ii) "false negative", i.e., an actual sentence boundary which the method did not label as such. Table 3 contains a summary of these errors.

These errors can be decomposed into the following groups:

> (37.6%) false positive at an abbreviation within a title or name, usually because the word following the period exists in the lexicon with other parts of speech (*Mr. Gray, Col. North, Mr. Major, Dr. Carpenter, Mr. Sharp*).
>
> (22.5%) false negative due to an abbreviation at the end of a sentence, most frequently *Inc., Co., Corp.*, or *U.S.*, which all occur within sentences as well.
>
> (11.0%) false positive or negative due to a sequence of characters including a period and quotation marks, as this sequence can occur both within and at the end of sentences.
>
> (9.2%) false negative resulting from an abbreviation followed by quotation marks; related to the previous two types.
>
> (9.8%) false positive or false negative resulting from presence of ellipsis (...), which can occur at the end of or within a sentence.
>
> (9.9%) miscellaneous errors, including extraneous characters (dashes, asterisks, etc.), ungrammatical sentences, misspellings, and parenthetical sentences.

The first two items indicate that the system is having difficulty recognizing the function of abbreviations. We attempted to counter this by dividing the abbreviations in the lexicon into two distinct categories, "title abbreviations" such as *Mr.* and *Dr.* which almost never occur at the end of a sentence, and all other abbreviations. This new classification, however, significantly increased the training time and eliminated only 12 of the 409 errors (2.9%).

The third and fourth items demonstrate the difficulty of distinguishing subsentences within a sentence. This problem may be addressed by creating a new classification for punctuation marks, the "embedded end-of-sentence," as suggested in Section 1. The fifth class of error may similarly be addressed by creating a new classification for ellipses, and then attempting to determine the role of the ellipses independent of the sentence boundaries.

Table 3
Results of testing on 27,294 mixed-case items; $t_0 = t_1 = 0.5$, 6-context, 2 hidden units.

224 (54.8%) false positives 185 (45.2%) false negatives
409 total errors out of 27,294 test cases

4.4 Lexicon size

The results in Sections 4.1-4.3 depended on a very large lexicon with more than 30,000 words. It is not always possible to obtain or build a large lexicon, so it is important to understand the impact of a smaller lexicon on the training time and error rate of the system. We altered the size of the English lexicon used in training and testing by removing large sections of the original lexicon and obtained the results in Table 4. These data demonstrate that a larger lexicon provides faster training and a lower error rate, although the performance with the smaller lexica was still almost as accurate. In the experiments described in Sections 4.5-4.10, we used a 5,000 word lexicon.

It is important to note, however, that in reducing the size of the lexicon as a whole, the number of abbreviations remained constant (at 206). Recognizing abbreviations gives important evidence as to the location of sentence boundaries, and reducing the number of abbreviations in the lexicon naturally reduces the accuracy of the system. Most existing boundary disambiguation systems, such as the STYLE program, depend heavily on abbreviation lists and would be relatively ineffective without information about abbreviations. However, the robustness of the Satz system allows it to still produce a relatively high accuracy without relying on extensive abbreviation lists. To demonstrate this robustness, we removed all abbreviations from the lexicon after reducing it in size to 5,000 words. The resulting Satz error rate was 4.9%, which was still significantly better than the STYLE baseline error rate of 8.3%, which was obtained with a 48 entry abbreviation list.

Table 4
Results of comparing lexicon size (27,294 potential sentence boundaries).

Words in Lexicon	Training Epochs	Testing Errors	Error (%)
30,000	218	411	1.5%
5,000	372	483	1.8%
3,000	1056	551	2.0%

4.5 Single-case results

A major advantage of the Satz approach to sentence boundary recognition is its robustness. In contrast to many existing systems which depend on brittle parameters such as capitalization and spacing, Satz is able to adapt to texts which are not well-formed, such as single-case texts. The two descriptor array flags for capitalization, discussed in Section 3.3, allow the system to include capitalization information when it is available. When this information is not available, the system is nevertheless able to adapt and produce a low error rate. To demonstrate this robustness, we converted the training, cross-validation, and test texts used in previous testing to a lower-case-only format, with no capital letters. After retraining the neural network with the lower-case-only texts, the Satz system was able to correctly disambiguate all but 3.3% of the sentence boundaries. After converting the texts to an upper-case-only format, with all capital letters, and retraining the network on the texts in this format, the system was able to correctly label all but 3.5%.[16]

4.6 Results on OCR texts

A very large and ever-increasing source of online texts is texts obtained via optical character recognition (OCR). These texts require very robust processing methods, as they contain a large number of extraneous and incorrect characters. The robustness results of the Satz system in the absence of an abbreviation list and capitalization suggest that it would be well suited for processing OCR texts as well. To test this, we prepared a small corpus of raw OCR data containing 1157 punctuation marks. The STYLE program

[16] The difference in results with upper-case-only and lower-case-only formats can probably be attributed to the capitalization flags in the descriptor arrays, as these flags would always be on in one case and off in the other.

produced an error rate of 11.7% over the OCR texts; the Satz system, using a neural network trained on mixed-case WSJ texts, produced an error rate of 4.2%.

In analyzing the sources of the errors produced by Satz over the raw OCR data, it was clear that many errors came from areas of high noise in the texts, such as the line in example (6), which contains an extraneous question mark and three periods. These areas probably represented charts or tables in the source text and would most likely need to be eliminated anyway, as it is doubtful any text processing program would be able to productively process them. We therefore applied a simple filter to the raw OCR data to locate areas of high noise and remove them from the text. In the resulting text of 1115 punctuation marks, the STYLE program had an error rate of 9.6% while the Satz system improved to 1.9%.

(6) e:)i. i)’e;y’,?;.i#i TCF. grades’ are’

(7) newsprint. Furthermore, shoe presses have Using rock for granite roll Two years ago we reported on advances in

While the low error rate on OCR texts is encouraging, it should not be viewed as an absolute figure. One problem with OCR texts is that periods in the original text may be scanned as commas or dropped from the text completely. Our system is unable to detect these cases. Similarly, the definition of a sentence boundary is not necessarily absolute, as large parts of texts may be incorrectly or incompletely scanned by the OCR program. The resulting “sentences” may not correspond to those in the original text, as can be seen in example (7). Such problems cause a low “error rate” to have less significance in OCR texts than in more well-formed texts such as the WSJ corpus.

4.7 Probabilistic vs. binary inputs

In the discussion of methods of representing context in Section 3.2.1, we suggested two ways of approximating the part-of-speech distribution of a word, using *prior probabilities* and *binary features.* The results reported in the previous sections were all obtained using the prior probabilities in the descriptor arrays for all tokens. Our experiments in comparing probabilistic inputs to binary feature inputs, given in Table 5, indicate that using binary feature inputs significantly improves the performance of the system on both mixed-case and single-case texts, as well as decreasing the training times. The lower error rate and faster training time suggest that the simpler approach of using binary feature inputs to the neural network is better than the frequency-based inputs previously used.

Table 5
Results of comparing probabilistic to binary feature inputs (5,000 word lexicon, 27,294 English test cases).

	Probabilistic			Binary		
	Training Epochs	Testing Errors	Error (%)	Training Epochs	Testing Errors	Error (%)
Mixed case	368	483	1.8%	312	474	1.7%
Lower case	182	890	3.3%	148	813	3.0%
Upper case	542	956	3.5%	190	744	2.7%

4.8 Thresholds

As described in Section 3.4.1, the output of the neural network (after passing through the sigmoidal squashing function) is used to determine the function of a punctuation mark based on its value relative to two sensitivity thresholds, with outputs that fall between the thresholds denoting that the function of the punctuation mark is still ambiguous. These are shown in the "Not Labeled" column of Table 6, which gives the results of a systematic experiment with the sensitivity thresholds. As the thresholds were moved from the initial values of 0.5 and 0.5, certain items which had been classified as "False Pos" or "False Neg" fell between the thresholds and became "Not Labeled." At the same time, however, items which had been correctly labeled also fell between the thresholds, and these are shown in the "Were Correct" column.[17] There is thus a tradeoff: decreasing the error percentage by adjusting the thresholds also decreases the percentage of cases correctly labeled and increases the percentage of items left ambiguous.

Table 6
Results of varying the sensitivity thresholds (27,294 test cases, 6-context, 2 hidden units).

Lower Thresh	Upper Thresh	False Pos	False Neg	Not Labeled	Were Correct	% Not Labeled	Error (%)
0.5	0.5	209	200	0	0	0.0	1.5
0.4	0.6	173	174	145	83	0.50	1.3
0.3	0.7	140	148	326	205	1.20	1.1
0.2	0.8	111	133	541	376	1.98	0.9
0.1	0.9	79	94	1021	785	3.74	0.6

4.9 Amount of training data

To obtain the results in Sections 4.1-4.8, we used very small training and cross-validation sets of 573 and 258 items, respectively. The training and cross-validation sets could thus be constructed in very few minutes, and the resulting system error rate was very low. To determine the system improvement with more training data, we removed a portion of the test data and incrementally added it to the training and cross-validation sets. We found that, after an initial increase in the error rate, which can probably be accounted for by the fact that the new training data came from a different part of the corpus, increasing the size of the training and cross-validation sets to 2514/1266 reduced the error percentage to 1.1%, as can be seen in Table 7. The tradeoff for this decreased error rate is a longer training time (often more than 10 minutes) as well as the extra time required to construct the larger sets.

Table 7
Results of varying size of training and cross-validation sets (19034 test items).

Training Items	Cross-validation Items	Training Epochs	Error (%)
573	258	85	1.5%
622	587	84	1.8%
1174	587	135	2.0%
1858	1266	172	1.2%
2514	1266	222	1.1%
3179	1952	316	1.3%

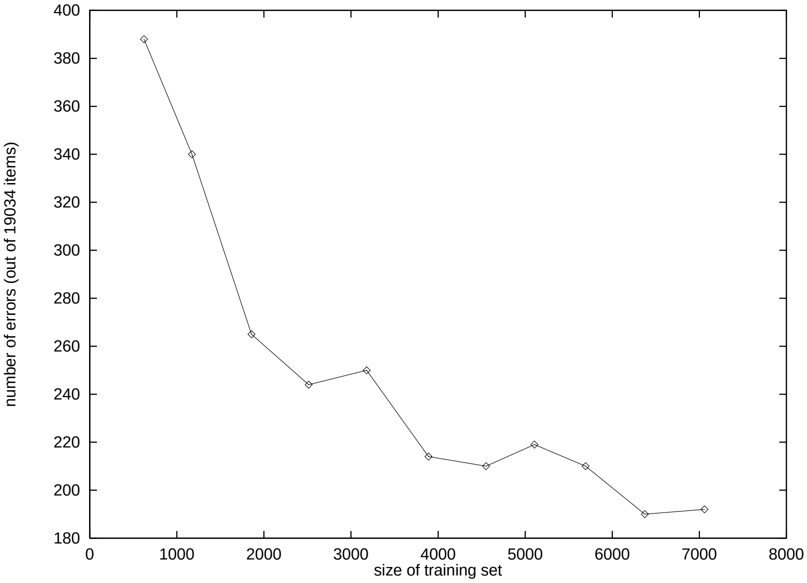
[17] Note that the number of items in the "Were Correct" column is a subset of those in the "Not Labeled" column.

4.10 Decision trees

We next compared the Satz system error rate obtained using the neural network with results using a decision tree. We were able to use some of the previous results, specifically the optimality of a 6-context and the effectiveness of a smaller lexicon and binary feature vectors, to obtain a direct comparison with the neural net results. We used the c4.5 decision tree induction program(Quinlan, 1993) and a 5,000 word lexicon to produce all decision tree results.

4.10.1 Size of training set. As we showed in Section 4.9, the size of the training set used for the neural network affected the overall system error rate. The same was true with the decision tree induction algorithm, as seen in Figure 4. The lowest error rate (1.0%) was obtained with a training set of 6373 items.

Figure 4
Results of increasing training set size for decision tree induction (19034 item test set).



4.10.2 Mixed-case results. One advantage of decision tree induction is that the algorithm clearly indicates which of the input attributes are most important. While the 6-context descriptor arrays present 120 input attributes to the algorithm, c4.5 induced a decision tree utilizing only 10 of the attributes, when trained on the same mixed-case WSJ text used to train the neural network. The 10 attributes for mixed-case English texts, as seen in the induced decision tree in Figure 5 are (where t-1 is the token preceding the punctuation mark, t+1 is the token following, and so on):

t-1 can be an abbreviation

t+1 is a comma or semicolon

t+1 can be a sentence-ending punctuation mark

t+1 can be a pronoun

t+1 is capitalized

t+1 can be a conjunction

t+1 can be a proper noun

t+2 can be a noun

t-3 can be a modifier

t-3 can be a proper noun

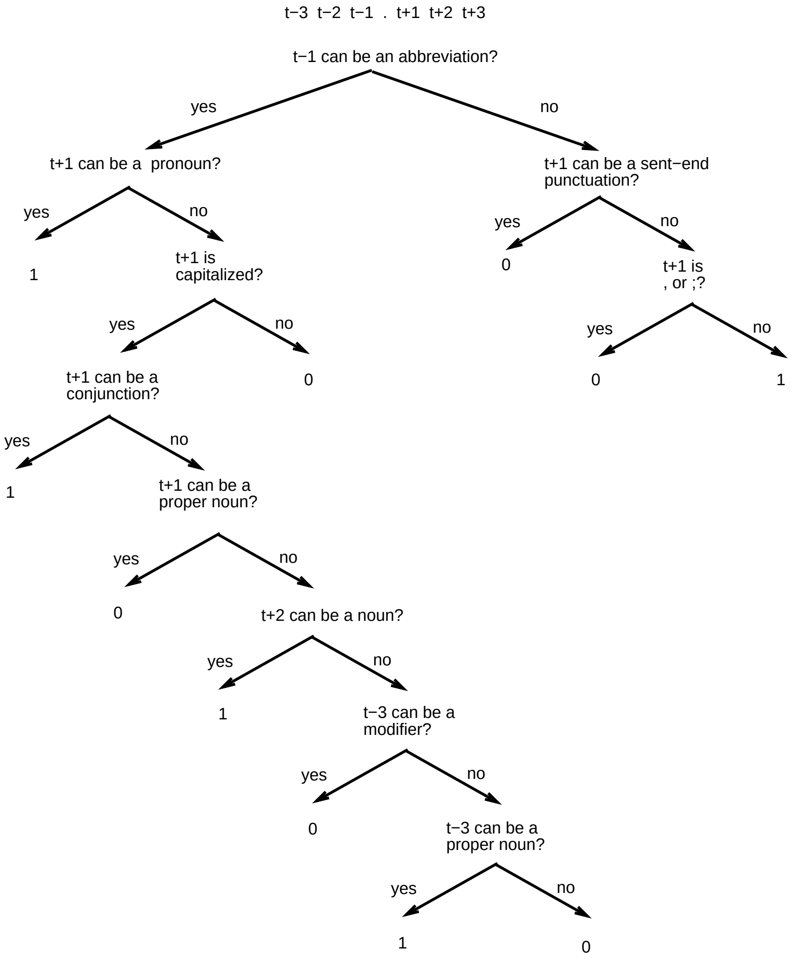

t−3 t−2 t−1 . t+1 t+2 t+3
t−1 can be an abbreviation?
yes
no
t+1 can be a pronoun?
yes
no
1
t+1 is capitalized?
yes
no
t+1 can be a conjunction?
yes
no
1
t+1 can be a proper noun?
yes
no
0
t+2 can be a noun?
yes
no
1
t−3 can be a modifier?
yes
no
0
t−3 can be a proper noun?
yes
no
1
0
0
t+1 can be a sent−end punctuation?
yes
no
0
t+1 is , or ;?
yes
no
0
1

Figure 5
Decision tree induced for mixed-case English texts. Leaf nodes labeled with 1 indicate that the punctuation mark is a sentence boundary.

The decision tree created from the small training set of 622 items resulted in an error rate of 1.6%. This result was slightly higher than the lowest error rate (1.5%) obtained with neural network trained with a similar training set and a 5,000 word lexicon. The lowest error rate obtained using a larger training set to induce the decision tree (1.0%), however, is better than the lowest error rate (1.1%) for the neural network trained on a larger set.

4.10.3 Single-case results. Running the induction algorithm on upper-case-only and lower-case-only texts both produced the same decision tree, shown in Figure 6. An interesting feature of this tree is that it reduced the 120 input attributes to just 4 important ones. Note the similarity of this tree to the algorithm used by the STYLE program as discussed in Section 2.

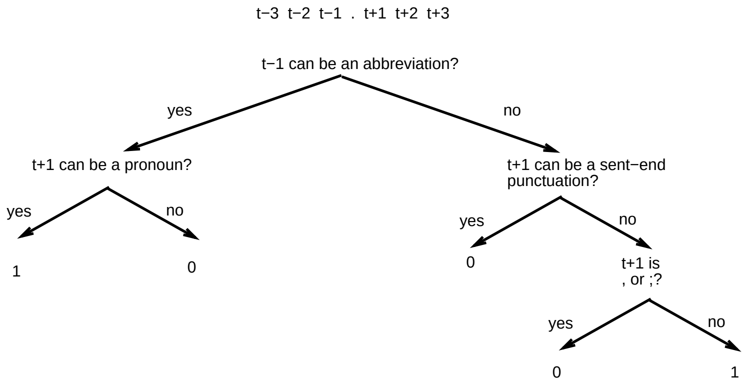

t−3 t−2 t−1 . t+1 t+2 t+3
t−1 can be an abbreviation?
yes
no
t+1 can be a pronoun?
yes
no
1
0
t+1 can be a sent−end punctuation?
yes
no
0
t+1 is , or ;?
yes
no
0
1

Figure 6
Decision tree induced for single-case English texts.

Trained on 622 items, this tree produced 527 errors over the 27,294 item test set, an error rate of 1.9%, for both upper-case-only and lower-case-only texts. This error rate is lower than the best result for the neural network (3.3%) on single-case texts, despite the small size of the training set used.

5. Adaptation to Other Languages

Since the disambiguation component of the sentence boundary recognition system, the learning algorithm, is language independent, the Satz system can be easily adapted to natural languages with punctuation systems similar to English. Adaptation to other languages involves obtaining (or building) a small lexicon containing the necessary part-of-speech data and constructing small training and cross-validation texts. We have successfully adapted the Satz system to German and French, and the results are described below.

5.1 German

The German lexicon was built from a series of public-domain word lists obtained from the Consortium for Lexical Research. In the resulting lexicon of 17,000 German adjectives, verbs, prepositions, articles, and 156 abbreviations, each word was assigned only the parts of speech for the lists from which it came, with a frequency of 1 for each part of speech. As the lack of actual frequency data in the lexicon made construction of a probabilistic descriptor array impossible, we performed all German experiments using binary vectors. The part-of-speech tags used were identical to those from the English lexicon, and the descriptor array mapping remained unchanged. This lexicon was used in testing with two separate corpora. The total development time required to adapt Satz to German, including building the lexicon and constructing training texts, was less than one day. We tested the system with two separate German corpora.

5.1.1 Süddeutsche Zeitung corpus. The Süddeutsche Zeitung corpus consists of several megabytes of on-line texts from the German newspaper.[18] We constructed a training

[18] All work with the Süddeutsche Zeitung corpus was performed in collaboration with the University of Munich.

text of 520 items from the Süddeutsche Zeitung corpus, and a cross-validation text of 268 items. Training was performed in less than 5 minutes on a Next workstation.[19] Testing on a sample of 3184 separate items from the same corpus resulted in error rates less than 1.3%, as summarized in Table 8. A direct comparison to the UNIX STYLE program is not possible for German texts, as the STYLE program is only effective for English texts. The SZ corpus did have a lower bound of 79.1%, which was similar to the 75.0% lower bound of the WSJ corpus.

5.1.2 German News corpus. The German News corpus was constructed from a series of public-domain German articles distributed internationally by the University of Ulm. We constructed a training text of 268 potential sentence boundaries from the corpus, as well as a cross-validation text of 150 potential sentence boundaries, and the training time was less than one minute in all cases. A separate portion of the corpus was used for testing the system and contained over 5,037 potential sentence boundaries from the months July-October 1994, with a baseline system performance of 96.7%. Results of testing on the German News corpus are given in Table 8 and show a very low error rate for both mixed-case and single-case texts. Repeating the testing with a smaller lexicon containing less than 2,000 words still produced an error rate lower than 1% with a slightly higher training time.

5.1.3 Decision tree induction for German texts. Using the c4.5 program to induce decision trees from a 788 item German training set (from the SZ corpus) resulted in a tree utilizing 11 of the 120 attributes. The error rates over the SZ test set were 2.3% for mixed-case texts and 1.9% for single-case texts, both noticeably higher than the best error rate (1.3%) achieved with the neural network on the SZ corpus. The decision tree induced for the German News corpus utilized only three attributes (t-1 can be an abbreviation, t-1 can be a number, t+1 can be a noun) and produced a 0.7% error rate in all cases.

Table 8
German results (17,000 word lexicon).

	Training Epochs	Testing Errors	Error (%)
Süddeutsche Zeitung mixed case	204	42	1.3%
lower case	141	44	1.4%
upper case	274	43	1.4%
Decision tree mixed		72	2.3%
single		61	1.9%
German News mixed case	732	37	0.7%
lower case	678	25	0.5%
upper case	611	36	0.7%
Decision tree mixed		36	0.7%
single		36	0.7%

5.2 French

The French lexicon was compiled from the part-of-speech data obtained by running the Xerox PARC part-of-speech tagger (Cutting et al., 1991) on a portion of the French half

[19] The Next workstation is significantly slower than the DEC Alpha workstation used in other tests, which accounts for the slower training time.

of the Canadian Hansards corpus. The lexicon consisted of less than 1000 words assigned parts of speech by the tagger, including 20 French abbreviations appended to the 200 English abbreviations available from the lexicon used in obtaining the results described in Section 4. The part-of-speech tags in the lexicon were different from those used in the English implementation, so the descriptor array mapping had to be adjusted accordingly. The development time required to adapt Satz to French was 2 days. A training text of 361 potential sentence boundaries was constructed from the Hansards corpus, and a cross-validation text of 137 potential sentence boundaries, and the training time was less than one minute in all cases. The results of testing the system on a separate set of 3766 potential sentence boundaries (also from the Hansards corpus) with a baseline algorithm performance of 80.1% are given in Table 9, including a comparison of results with both probabilistic and binary feature inputs. These data show a very low system error rate on both mixed and single-case texts. In addition, the decision tree induced from 498 French training items by the c4.5 program produced a lower error rate (0.4%) in all cases.

Table 9
French results (1,000 word lexicon, 3,766 potential sentence boundaries).

	Probabilistic			Binary		
	Training Epochs	Testing Errors	Error (%)	Training Epochs	Testing Errors	Error (%)
Mixed case	273	39	1.0%	302	22	0.6%
Lower case	243	24	0.6%	181	29	0.8%
Upper case	328	25	0.7%	223	21	0.6%
Decision tree (all cases)					17	0.4%

6. Improving Performance on the Difficult Cases

To demonstrate the performance of our system within a large-scale NLP application, we integrated the Satz system into an existing information extraction system, the Alembic system (Aberdeen et al., 1995) described in Section 2.2. On the same WSJ corpus used to test Satz in Section 4, the Alembic system alone achieved an error rate of only 0.9% (the best error rate achieved by Satz was 1.0%). A large percentage of the errors made by the Alembic module fell into the second category described in Section 4.3, where one of the five abbreviations *Co., Corp., Ltd., Inc.*, or *U.S.* occurred at the end of a sentence. We decided to see if Satz could be applied in such a way that it improved the results on the hard cases on which the hand-written rules were unable to perform as well as desired.

We trained Satz on 320 of the problematic examples listed above, taken from the WSJ training corpus. The remaining 19,034 items were used as test data. The Alembic module was used to disambiguate the sentence boundaries in all cases except when one of the five problematic abbreviations was encountered; in these cases, Satz (in neural network mode) was used to determine the role of the period following the abbreviation. The hybrid disambiguation system reduced the total number of sentence boundary errors by 46% and the error rate for the whole corpus fell from 0.9% to 0.5%.

We trained Satz again, this time using the decision tree learning method, in order to see what kind of rules were acquired for the problematic sentences. The decision tree shown in Figure 7 is the result; the performance of Satz with this decision tree was nearly identical to the performance with the neural network. From this tree it can be seen that context farther away from the punctuation mark is important, and extensive use is made of part-of-speech information.

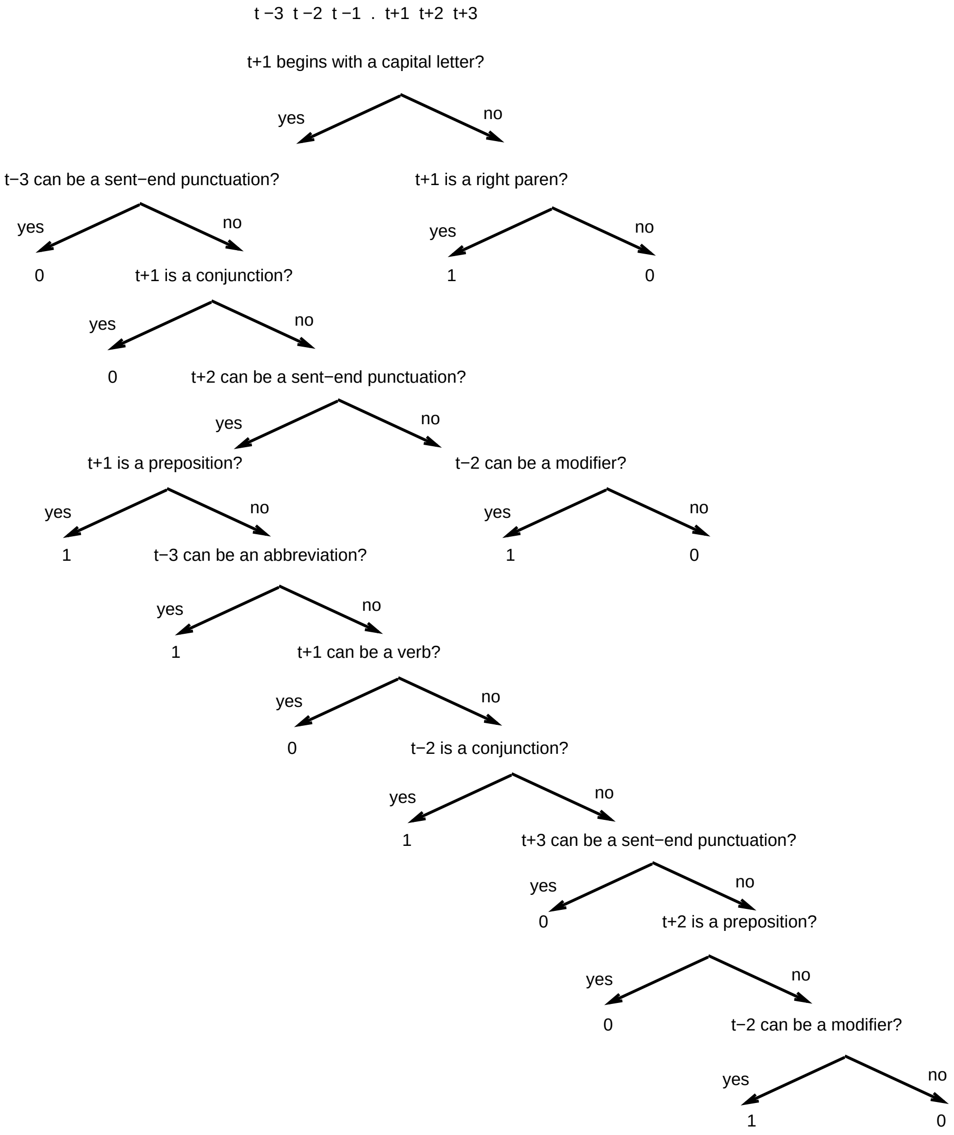


Figure 7
Decision tree induced when training on difficult cases exclusively.

7. Summary

This paper has presented Satz, a sentence boundary disambiguation system that possesses the following desirable characteristics:

- It is robust, and does not require a hand-built grammar or specialized rules that depend heavily on capitalization, multiple spaces between sentences, etc.
- It adapts easily to new text collections.
- It adapts easily to new languages.
- It trains and executes quickly without requiring large resources.

- It produces very accurate results.
- It is efficient enough that it does not noticeably slow down text preprocessing.
- It is able to specify "no opinion" on cases that are too difficult to disambiguate, rather than making under-informed guesses.

The Satz system offers a robust, rapidly trainable alternative to existing systems, which usually require extensive manual effort and are tailored to a specific text genre or a particular language. By using part-of-speech frequency data to represent the context in which the punctuation mark appears, the system offers significant savings in parameter estimation and training time over word-based methods, while at the same time producing a very low error rate (see Table 10 for a summary of the best results for each language).

The systems of Riley (1989) and Wasson report what seem to be slightly better error rates, but these results are not directly comparable since they were evaluated on other collections. Furthermore, the Wasson system required 9 staff months of development, and the Riley system required 25 million word tokens for training and storage of probabilities for all corresponding word types. By comparison, the Satz approach has the advantages of flexibility for application to new text genres, small training sets (and thereby fast training times), relatively small storage requirements, and little manual effort. The training time on a workstation (in our case a DEC Alpha 3000) is less than one minute, and the system can perform the sentence boundary disambiguation at a rate exceeding 10,000 sentences/minute. Because the system is lightweight, it can be incorporated into the tokenization stage of many natural language processing systems without substantial penalty. For example, combining our system with a fast sentence alignment program such as that of (Gale and Church, 1993), which performs alignment at a rate of up to 1000 sentences/minute, would make it possible to rapidly and accurately create a bilingual aligned corpus from raw parallel texts. Because the system is adaptive, it can be focused on especially difficult cases and combined with existing systems to achieve still better error rates, as shown in Section 6.

The system was designed to be easily portable to new natural languages, assuming the accessibility of lexical part-of-speech information. The lexicon itself need not be exhaustive, as shown by the success of adapting Satz to German and French with limited lexica, and by the experiments in English lexicon size described in Section 4.4. The heuristics used within the system to classify unknown words can compensate for inadequacies in the lexicon, and these heuristics can be easily adjusted.

It is interesting that the system performs so well using only estimates of the parts of speech of the tokens surrounding the punctuation mark, and using very rough estimates at that. In future it may be fruitful to apply a technique that uses such simple information to more complex problems.

Table 10
Summary of best results.

Corpus	Size	Lower Bound	Baseline (STYLE)	Satz Neural Net	Satz Decision Tree
Wall Street Journal	27,294	75.0	8.3%	1.1%	1.0%
Süddeutsche Zeitung	3,184	79.1		1.3%	1.9%
German News	5,037	96.7		0.7%	0.7%
Hansards (French)	3,766	80.1		0.6%	0.4%

Acknowledgments

The authors would like to acknowledge valuable advice, assistance, and encouragement provided by Manuel Fähndrich, Haym Hirsh, Dan Jurafsky, and Terry Regier, and the four anonymous reviewers whose comments have greatly improved this paper. We would also like to thank Ken Church for making the PARTS data available, and Ido Dagan, Christiane Hoffmann, Mark Liberman, Jan Pedersen, Martin Röscheisen, Mark Wasson, and Joe Zhou for assistance in finding references and determining the status of related work.

References

Aberdeen, John, John Burger, David Day, Lynette Hirschman, Patricia Robinson, and Marc Vilain. 1995. MITRE: Description of the Alembic system used for MUC-6. In *The Proceedings of the Sixth Message Understanding Conference (MUC-6)*, Columbia, Maryland, November 1995.

Bahl, L.R., P.F. Brown, P.V. deSouza, and R.L. Mercer. 1989. A tree-based statistical language model for natural language speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(7):1001–1008.

Bourland, Hervé and Nelson Morgan. 1994. *Connectionist Speech Recognition: A Hybrid Approach*. Kluwer Academic Publishers, Norwell, Mass.

Breiman, Leo, Jerome H. Friedman, Richard Olshen, and Charles J. Stone. 1984. *Classification and regression trees*. Wadsworth International Group, Belmont, CA.

Cherry, L.L. and W. Vesterman. 1991. Writing tools – the STYLE and DICTION programs. In *4.3BSD UNIX System Documentation*. University of California, Berkeley

Church, Kenneth W. 1988. A stochastic parts program and noun phrase parser for unrestricted text. In *Second Conference on Applied Natural Language Processing*, pages 136–143, Austin, TX.

Church, Kenneth W. 1993. Char align: A program for aligning parallel texts at the character level. In *31st Annual Meeting of the Association for Computational Linguistics*, pages 1–8, Columbus, OH.

Church, Kenneth W. and Mark Y. Liberman. 1991. A status report on the ACL/DCI. In *The Proceedings of the 7th Annual Conference of the UW Centre for the New OED and Text Research: Using Corpora*, pages 84–91, Oxford.

Cutting, Doug, Julian Kupiec, Jan Pedersen, and Penelope Sibun. 1991. A practical part-of-speech tagger. In *The 3rd Conference on Applied Natural Language Processing*, Trento, Italy.

Francis, W. and H. Kucera. 1982. *Frequency Analysis of English Usage*. Houghton Mifflin Co., New York.

Fung, Pascale and Kenneth Ward Church. 1994. K-vec: A new approach to aligning parallel texts. In *Proceedings of COLING-94*.

Fung, Pascale and Kathleen McKeown. 1994. Aligning noisy parallel corpora across language groups: Word pair feature matching by dynamic time warping. In *Proceedings of AMTA-94*.

Gale, William A. and Kenneth W. Church. 1993. A program for aligning sentences in bilingual corpora. *Computational Linguistics*, 19(1):75–102.

Hertz, John, Anders Krogh, and Richard G. Palmer. 1991. *Introduction to the theory of neural computation*. Santa Fe Institute studies in the sciences of complexity. Addison-Wesley Pub. Co., Redwood City, CA.

Hoffmann, Christiane. 1994. Automatische Disambiguierung von Satzgrenzen in einem maschinenlesbaren deutschen Korpus. Unpublished work with Prof. R. Köhler at University of Trier, Germany.

Humphrey, T.L. and F.-q. Zhou. 1989. Period disambiguation using a neural network. In *IJCNN : International Joint Conference on Neural Networks*, page 606, Washington, D.C.

Kay, Martin and Martin Röscheisen. 1993. Text-translation alignment. *Computational Linguistics*, 19(1):121–142.

Lesk, M. E. and E. Schmidt. 1975. Lex - a lexical analyzer generator. Computing Science Technical Report 39, AT&T Bell Laboratories, Murray Hill, N.J.

Liberman, Mark Y. and Kenneth W. Church. 1992. Text analysis and word pronunciation in text-to-speech synthesis. In Sadaoki Furui and Man Mohan Sondhi, editors, *Advances in Speech Signal Processing*. Marcel Dekker, Inc., pages 791–831.

Lippmann, R. P. 1989. Review of neural networks for speech recognition. *Neural Computation*, 1:1–38.

Magerman, David M. 1995. Statistical decision-tree models for parsing. In *Proceedings of the 33rd Meeting of the*

Association for Computational Linguistics.

Melamed, I. Dan. 1995. Automatic evaluation and uniform filter cascades for inducing n-best translation lexicons. In *Proceedings of the Workshop on Very Large Corpora 1995.*

Mitamura, Teruko, Eric H. Nyberg, and Jaime G. Carbonell. 1993. Automated corpus analysis and the acquisition of large, multi-lingual knowledge bases for MT. In *5th International Conference on Theoretical and Methodological Issues in Machine Translation*, Kyoto, Japan.

Müller, Hans, V. Amerl, and G. Natalis. 1980. Worterkennungsverfahren als Grundlage einer Universalmethode zur automatischen Segmentierung von Texten in Sätze. Ein Verfahren zur maschinellen Satzgrenzenbestimmung im Englischen. *Sprache und Datenverarbeitung*, 1.

Nakamura, Masami, Katsuteru Maruyama, Takeshi Kawabata, and Kiyohiro Shikano. 1990. Neural network approach to word category prediction for english texts. In *Proceedings of COLING-90*, volume 3, pages 213–218.

Nicol, G. T. 1993. *Flex - The Lexical Scanner Generator.* The Free Software Foundation, Cambridge, MA.

Nunberg, Geoffrey. 1990. *The Linguistics of Punctuation.* C.S.L.I. Lecture Notes, Number 19. Center for the Study of Language and Information, Stanford, CA.

Palmer, David D. 1994. Satz - an adaptive sentence segmentation system. M.S. Thesis and UC-Berkeley Technical Report UCB/CSD 94-846.

Palmer, David D. and Marti A. Hearst. 1994. Adaptive sentence boundary disambiguation. In *Proceedings of the Fourth ACL Conference on Applied Natural Language Processing*, pages 78–83. Morgan Kaufmann, Oct.

Quinlan, J.R. 1986. Induction of decision trees. *Machine Learning*, 1(1):81–106.

Quinlan, J.R. 1993. *c4.5: Programs for Machine Learning.* Morgan Kaufman, San Mateo, CA.

Resnik, Philip. 1993. Semantic classes and syntactic ambiguity. In *Proceedings of the ARPA Workshop on Human Language Technology (March 1993).*

Riley, Michael D. 1989. Some applications of tree-based modelling to speech and language indexing. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 339–352. Morgan Kaufmann.

Schmid, Helmut. 1994. Part-of-speech tagging with neural networks. In *Proceedings of COLING-94.*

Siegel, Eric V. and Kathleen R. McKeown. 1994. Emergent linguistic rules from inducing decision trees: Disambiguating discourse clue words. In *Proceedings of AAAI 1994*, pages 820–826. AAAI Press/MIT Press, July.

Simard, Michel, George F. Foster, and Pierre Isabelle. 1992. Using cognates to align sentences in bilingual corpora. In *Proceedings of TMI92.*

Soderland, Stephen and Wendy Lehnert. 1994. Corpus-driven knowledge acquisition for discourse analysis. In *Proceedings of AAAI 1994*, pages 827–832. AAAI Press/MIT Press, July.

Wall, Larry and Randal L. Schwartz. 1991. *Programming perl.* O'Reilly and Associates, Inc., Sebastopol, CA.