# Using single layer networks for discrete, sequential data: an example from natural language processing
Extended version

Caroline Lyon
School of Information Sciences
University of Hertfordshire
Hatfield, Herts. AL10 9AB
email: C.M.Lyon@herts.ac.uk
Tel: +44 (0)1707 284266

April 12, 1996

Comp. Sci
Tech No:
242

## Abstract

Supervised, feed-forward networks will, in general, need more than one layer to process data. However, if they can be used, single layer networks offer advantages of functional transparency and operational speed. Now, in some cases data can be pre-processed and then presented in a linearly separable form for processing by a single layer net. In effect, processing at different stages can be de-coupled. The critical issue is finding the pre-processing function to convert data into an appropriate form. For characteristic linguistic data this can be done, and a natural language parser which has been successfully implemented is used to investigate the approach.

Single layer nets can then be trained by finding weight adjustments based on (a) factors proportional to the input, as in the Perceptron, (b) factors proportional to the existing weights, and (c) an error minimization method. In our experiments generalization ability varies little; method (b) has been used for the prototype parser.

## 1 Introduction

This paper examines some of the issues that have to be addressed in designing neural processors for discrete, sequential data. There is a mutual dependence between the representation of data on the one hand and, on the other, the architecture and function of an effective network. As a vehicle for examining these processes we take an automated partial parser that has been successfully developed [1, 2]. [1] Ability to generalize is the primary concern.

In principle, simpler networks with well understood functions have *prima facie* advantages, so looking for a representation that enables such networks to be used should be advantageous. With feed forward, supervised networks single layer models enjoy functional transparency and operational speed, but in general this type of network will need more than one layer to process data. More complex models such as the multi-layer Perceptron are commonly used. Their training algorithms integrate weight adjustments at different layers: changes to connection weights at one layer are dynamically limked to those at another.

However, there is an alternative approach. The layers may be de-coupled, and processing at different layers done in separate steps. Data can be pre-processed, which is analogous to processing at the first layer, and then presented in a linearly separable form to a single layer net, which is analogous to to the second layer. This is illustrated in Figure 5, which shows in simplified form an archetype of the class of Generalized Single Layer Networks (GSLN). A number of different network types that belong to this class are listed by Holden and Rayner [3]. The critical issue is finding a pre-processing function to convert data into an appropriate, linearly separable, form.

In order to develop any neural solution to a problem the data must first be examined, and a representation devised that captures some of its structure. Then, if the advantages afforded by simple single layer nets are sought, factors such as the linear separability of this converted data, and patterns of distribution, need to be investigated. Processors of discrete, categorial data may handle material with distinctive characteristics that contrast with other types of data, and these characteristics may affect design decisions. Natural language processing highlights the need for preliminary investigation in these fields.

This paper describes how characteristic linguistic data can be usefully converted into a linearly separable representation that partially captures sequential form. Then it is processed by three different single layer nets, whose performance is compared. All three are feed forward models with supervised training. Connection weights can be found by adjustments based on (a) factors proportional to the input (b) factors proportional to the existing weights, and (c) factors related to the difference between desired and actual output, an error minimization method. Model (a) is a traditional Perceptron; model (b) is based on the Hodyne network introduced by Wyard and Nightingale at British Telecom Laboratories [4]; model (c) comes from the class of networks that use an LMS (Least Mean Square error) training algorithm. There is little difference in generalization ability, but network (b) performs slightly better and has been used for the parsing task undertaken here.

### Natural language processing

The automatic parsing of natural language poses a significant problem, and neural computing techniques can contribute to its solution. For an overview of the scope for work in this field see [5], or the IBM/Lancaster publication [6], section headed "The Deplorable State of the Art". Our prototype gives results of over 90% correct on declarative sentences from technical manuals

[1] To try the parser on your own text, via telnet, contact the author.

(see Section 9). Automated syntactic analysis of natural language has, in the last decade, been characterised by two paradigms. Traditional AI, rule based methods contrast with probabilistic approaches, in which stochastic models are developed from large corpora of real texts. Neural techniques fall into the broad category of the latter, data driven methods, with trainable models developed from examples of known parses. The parser we have implemented uses a hybrid approach: rule based techniques are integrated with neural processors.

Parsing can be taken as a pattern matching task, in which a number of parses are postulated for some text. A classifier distinguishes between the desired parse and incorrect structures. The pattern matching capabilities of neural networks have a particular contribution to make to the process, since they can conveniently model negative as well as positive relationships. The occurrence of some words or groups of words inhibit others from following, and these constraints can be exploited. Arguments on the need for negative information in processing formal languages [7, 8] can be extended to natural language. This is an important source of information which has been difficult for traditional probabilistic methods to access [9, 10]. Neural methods also have the advantage that training is done in advance, so the run time computational load is low.

### Contents of paper

This paper will first take an overview of some factors that are relevant to neural net design decisions, (Section 2). To develop the argument in this paper, on the importance of tailoring the processor to the problem, it is then necessary to look in some detail at characteristics of natural language (Section 3), and the representation of sequential data (Section 4). The paper explains briefly how the pattern matching capabilities of neural networks have been used in our automated system that partially parses natural language text, (Section 5). After describing the partial parser we examine some of the design issues for the neural components of the system. First, the data itself is examined closely (Section 6). Then we consider how the data can be mapped onto a representation suitable for processing with a single layer net (Section 7). Section 8 gives an overview of the design and training methods of the three different networks: (a) the Perceptron, (b) Hodyne and (c) an LMS model. Details of the training algorithms are in the Appendix. In Section 9 we compare the performance of the three networks, their ability to generalize. Training times are also recorded. Generalization is good, providing that the ratio of testing to training data is low enough. Over 90% of the data is correctly classified, and the output can be interpreted so that results for the practical application are up to 100% correct. On the small amount of data processed so far the different networks have roughly comparable generalization ability, but the Hodyne model is slightly better.

We conclude (Section 10) that linguistic data is a suitable candidate for processing with this approach. It is generally worth investigating ways of developing neural processors along the GSLN paradigm, so that the advantages of functional transparency and operational speed can be enjoyed.

## 2 Neural nets as classifiers of different types of data

Minsky and Papert describe their influential book "Perceptrons" [11] as a work that is concerned "with concepts that could help us to see the relation between patterns and the types of parallel machine architectures that might or might not be able to recognise them". This is also the purpose of our paper. In their critique Minsky and Papert closely examined the linear separability of certain sets of data, and its relationship to pattern learnability in a single layer net. In this section we identify other data characteristics that may be relevant to the design of appropriate classifiers.

## 2.1 "Clean" and noisy data

Let us first consider the fundamental difference in purpose between systems that handle noisy data, where it is desired to capture an underlying structure and smooth over some input, compared to those that process "clean" data, where every input datum may count. The many applications of neural nets in areas such as image processing provide examples of the first type, the parity problem is typical of the second. [2] These "clean" and "noisy" types can be considered as endpoints of a spectrum, along which different processing tasks lie. In the case of noisy data a classifier will be required to model significant characteristics in order to generalize effectively. The aim is to model the underlying function that generates the data, so the training data should not be over fitted. Bishop [12, chapter 9] gives a good account of principled methods that address this issue.

On the other hand, for types of data such as inputs to a parity detector, no datum is noise. Consider an input pattern that is markedly different, that is topologically distant, from others in its class. For one type of data this may be noise. In other instances an "atypical" vector may not be noise, and we may need to capture the information it carries to fix the class boundary effectively.

As we demonstrate in the next section, linguistic data needs to be analysed from both angles. We need to capture the statistical information on probable and improbable sequences of words; we also need to use the information from uncommon exemplars, which make up a very large proportion of natural language data.

## 2.2 Preserving topological relationships

Another of the characteristics that is relevant to network design is the extent to which the classification task, the mapping from input to output, preserves topological relationships. In some cases data which are close in input space are likely to produce the same output, and conversely similar classifications are likely to be derived from similar inputs. Optical character recognition is an example. However, there are other classification problems which are different: a very small change in input may cause a significant change in output and, on the other hand, very different input patterns can belong to the same class. Again, the parity problem is a paradigm example: in every case changing a single bit in the input pattern changes the desired output.

## 2.3 Data distribution and structure

Underlying data distribution and structure have their effect on the appropriate type of processor, and these characteristics should be examined. Information about the structure of linguistic data can be used to make decisions on suitable representations. In this work information theoretic techniques are used to support decisions on representation. We examine characteristics of linguistic data, and their influence on questions of network design is then shown.

Some recent work on the theory of generalization investigates the VC dimension, a measure of the expressive power of a classifier [13]. The probability of generalization error can be seen as a function of 2 factors only: the number of training examples and the level to which the

[2]The classical parity problem is to take a binary input vector and classify it as having an even or odd number of 1's.

training set is learnt. Arbitrary distribution of data is assumed. However, under this assumption the number of training examples required to bring the probability of error within the required bound may be impractically high. In practice we may use information on data distribution to improve generalization ability. As shown in Section 3, assumptions of normality cannot be made for linguistic data. The distribution indicates that in order to generalize adequately the processor must capture information from a significant number of infrequent events.

## 3 Characteristics of linguistic data

The significant characteristics of natural language that we wish to capture include:

- An indefinitely large vocabulary
- The distinctive distribution of words and other linguistic data
- A hierarchical syntactic structure
- Both local and distant dependencies, such as feature agreement

### 3.1 Vocabulary size

Shakespeare is said to have used a vocabulary of 30,000 words, and even an inarticulate computer scientist might need 15,000 to get by (counting different forms of the same word stem separately). Current vocabularies for speech processing databases, which include large sets of proper nouns, reach $O(10^5)$. Without specifying an upper limit we wish to be able to model an indefinite number of words.

### 3.2 The distinctive distribution of words and other linguistic data

The distribution of words in English and other languages has distinctive characteristics, to which Shannon drew attention [14, 1951]. Statistical studies were made of word frequencies in English language texts. In about 250,000 words of text (word tokens) there were about 30,000 different words (word types), and the frequency of occurrence of each word type was recorded. If word types are ranked in order of frequency, and $n$ denotes rank, then there is an empirical relationship between the probability of the word at rank $n$ occurring, $p(n)$, and $n$ itself. This is known as Zipf's Law [15]:

$$p(n) * n = constant$$

This gives a surprisingly good approximation to word probabilities in English and other languages, and indicates the extent to which a significant number of words occur infrequently. Dunning's analysis of the non-normal distribution of linguistic data illustrates this [16, 1993]. Words that have a frequency of less than 1 in 50,000 make up about 20-30% of typical English language news-wire reports. The LOB corpus [3] with about 1 million word tokens contains about 50,000 different word types, of which about 42,000 occur less than 10 times each [17]. In the Brown corpus of comparable size 40% of word types are *hapax legomena*, occurring only once [18].

[3]The London Oslo Bergen corpus and the Brown corpus are collections of text much used as raw material for natural language processing

The fact that a large number of words have a small probability of occurrence leads to processing problems. Consider, for instance, the well used trigram language model, which represents language as linear sequences of word triples. It is based on the assumption that the transition probability, the probability that one word will follow another, can be found. However, very large corpora of training data do not provide enough information to estimate transition probabilities satisfactorily. A report from IBM describes how a corpus of 1.5 million words of patent descriptions was analysed to find trigrams. When tested on 300,000 words from another part of the same corpus it was found that 23% of the trigrams in the test set had never occurred in the training set [19, 1992]. These figures indicate how sparse the data is likely to be. Even a corpus of 60 million words is considered too small to estimate trigram parameters reliably [20, 1992].

#### Mapping words onto part-of-speech tags

In order to address the problem of sparse data the vocabulary can be partitioned into groups, based on a similarity criterion, as is done in our system. An indefinitely large vocabulary is mapped onto a limited number of part-of-speech tag classes. This also make syntactic patterns more pronounced. Devising optimal tagsets is a significant task, on which further work remains to be done. For the purpose of this paper we take as given the tagsets used in the demonstration prototype, described in [21]. At the stage of processing described here 19 tags are used.

The "zipfian" distribution of words has been found typical of other linguistic data [22]. It is found again in the data derived from part-of-speech tags used to train the prototype described here: see Figures 3 and 4.

### 3.3 Hierarchical structure

There is an underlying grammatical structure to all natural languages, a phenomenon that has been extensively explored. Sentences will usually conform to certain structural patterns, as is shown in a simplified form in Figure 1. This is not inconsistent with the fact that acceptable grammatical forms evolve with time, and that people do not always express themselves grammatically.

Text also, of course, contains non sentential elements such as headings, captions, tables of contents. The work described in this paper is restricted to declarative sentences.

Note that the structure shown in Figure 1 is more general than a common approach which characterises basic sentence structure as

$$S \rightarrow NP \quad VP$$

meaning that a sentence can be rewritten as noun phrase followed by a verb phrase. As well as being used in traditional phrase structure grammars this approach has also commonly been adopted in connectionist language processing [23, 24, 25, 26]. A cursory glance at almost any real text will indicate that a phrase or clause preceding the subject is common. Statistics on the text used for our work are shown in Figure 2.

Within the grammatical structure there is an indefinite amount of variation in the way in which words can be assembled: acceptable sentences can be constructed from different combinations of words. On the other hand, the absence or presence of a single word can make a sentence unacceptable, for example

*There are many problems arise. .....(1)

*We is late. .....(2)

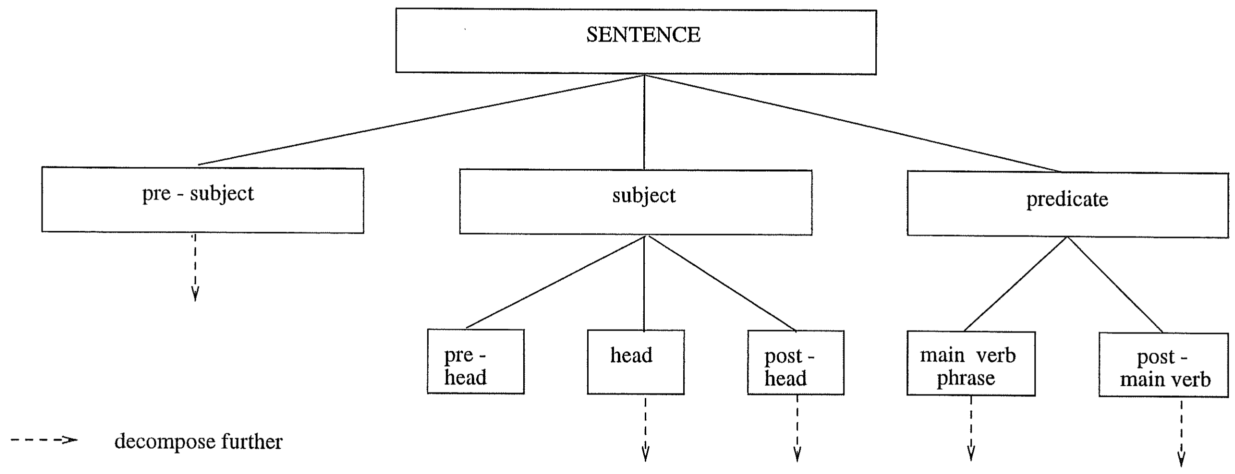

Figure 1: Decomposition of the sentence into syntactic constituents. Note that only the main verb phrase is mandatory. The other constituents are optional: for instance, an imperative sentence (such as the preceding sentence) has no explicit subject.

Now consider how linguistic data fits into the scheme described in Section 2.2 on preserving topological relationships. Many strings of words that are close in input space are also in the same grammatical category, but conversely on occasions a single word change can put a string into a different category. Our processor has to model this.

### 3.4 Local and distant dependencies

In examining natural language we find there are dependencies between certain words, both locally and at a distance. For instance, a plural subject must have a plural verb, so a construction like sentence (2) above is incorrect. This type of dependency in its general form is not necessarily local. In sentence (3) below the number of the subject is determined by its head, which in this type of example is not adjacent to the verb:

> If a cooler is fitted to the gearbox, [ the pipe [ connections ] of the cooler ] must be regularly checked for corrosion. ...(3)

The subject of this sentence is the plural "connections". Note that modal verbs like "must" have the same singular and plural form in English, but not in many other languages. For an automated translation system to process modal verbs it is necessary to find the head of the subject that governs the verb and ensure number agreement.

There are also dependencies between sentences and between more distant parts of a text. We aim to model just the intra-sentential dependencies as our automatic parser is developed.

## 4 Modelling sequential data

### 4.1 The moving window

Three methods have commonly been used to model sequential data, such as language, for connectionist processing. The first is to move a window along through the sequence, and process a series of static “snapshots”. Within each window ordering information is not represented. Sejnowski’s NETtalk is a well known example, among others [27, 28, 29].

### 4.2 Recurrent networks

Another method that shows promise and warrants further investigation is the use of recurrent nets [24, 30]. This would intuitively map naturally onto natural language. In its basic form this type of network is equivalent to a finite state automaton that can model regular languages [31]. However, the problem of modelling languages higher in the Chomsky hierarchy might be addressed either by decomposing the parsing task into appropriate subtasks, or by modifying the network [32], or both. Here again the data representation and the network must be reconciled.

### 4.3 The n-gram method

The third method, used in this work, is to take sets of ordered, adjacent elements, which captures some of the sequential structure of language. Combining tags into higher order tuples can also act as a pre-processing function, making it more likely that the resulting data can be processed by a single layer network (Section 7).

This method of representation captures some of the structure of natural language, as is shown by analysis with information theoretic techniques. There are relationships between neighbouring words in text: some are likely to be found adjacent, others are unlikely. When words are mapped onto part-of-speech tags this is also the case. This observation is supported by an investigation of entropy levels in the LOB corpus, in which 1 million words have been manually tagged.

Entropy can be understood as a measure of uncertainty [33, chapter 2]. The uncertainty about how a partial sequence will continue can be reduced when statistical constraints of neighbouring elements are taken into account. Shannon introduced this approach, see for instance his paper on “Prediction and Entropy of Printed English” [14, 1951]. He analysed sequences of letters, with the sequences represented by single letters, adjacent pairs and triples, with order preserved. These are n-grams, with $n$ equal to 1, 2 or 3. He found that the entropy of a sequence represented by letter n-grams declines as $n$ increases. When sequences of tags in the LOB corpus were analysed the same result was obtained: the entropy of part-of-speech n-grams declines as $n$ increases from 1 to 3.

This indicates that some of the structure of language is captured by taking tag pairs and triples as processing elements. It supports our decision to represent natural language strings as ordered sets of neighbouring tags. The n-gram method of representation has been much used in probabilistic processing. It was introduced for connectionist processing by Wyard and Nightingale [4].

We adopt the common approach of presenting data as binary vectors for all the networks examined in this work. Each element of the input vector represents an ordered tuple of adjacent part-of-speech tags, a pair or a triple. If a given tag tuple is present in an input string, then that element in the input vector is flagged to 1, else it remains 0.

## 5 The hybrid natural language processor

In order to process unrestricted natural language it is necessary to attack the problem on a broad front, and use every possible source of information. In our work the neural networks are part of a larger system, integrated with rule based modules, but they play a crucial role. We first *assert* that there is a syntactic structure which can be mapped onto a sentence (Figure 1). Then we use neural methods to find the mapping in each particular case. The grammar used is defined in [21].

## 5.1 Problem decomposition

In order to effect the mapping of this structure onto actual sentences we decompose the problem into stages, finding the boundaries of one syntactic feature at a time. Thus, the first step is to find the correct placement for the boundaries of the subject. Then further features are found in the 3 basic constituents. In the current prototype the head of the subject is subsequently identified. The processing at each stage is based on similar concepts, and to explain the role of the neural networks we shall in this paper discuss the first step in which the subject is found.

The underlying principle employed each time is to take a sentence, or part of a sentence, and generate strings with the boundary markers of the syntactic constituent in question placed in all possible positions. Then a neural net selects the string with the correct placement. The model is trained in supervised mode on marked up text to find this correct placement.

The performance of three different networks were compared: Perceptron, Hodyne and LMS. These different networks shared the same input and output routines, and each was integrated into the same larger system, which is outlined in this section.

## 5.2 Tagging

The stages in both the training and testing process start in the same way. The first decision on data representation is to map an indefinite number of words onto a limited number of part-of-speech tags. An automatic tagger allocates one or more part-of-speech tags to the words to be processed. Many words, typically perhaps 25% to 30%, have more than one tag. The CLAWS automatic tagger [34] provided a set of candidate tags for each word, but the probabilistic disambiguation modules were not used: disambiguating the tags is a sub-task for the neural processer. The CLAWS tagset was mapped onto a customised tagset of 19, used for the work described here. Further information on tagset development can be found in [21], and, briefly, in [1].

## 5.3 Hypertags as boundary markers

As well as part of speech tags we also introduce the syntactic markers, virtual tags, which at this stage of the process will demarcate the subject boundary. These *hypertags*, represent the opening '[' and closing ']' of the subject. The hypertags have relationships with their neighbours in the same way that ordinary tags do: some combinations are likely, some are unlikely. The purpose of the parser is to find the correct location of the hypertags.

With a tagset of 19 parts-of-speech, a start symbol and 2 hypertags we have 22 tags in all. Thus, there are potentially $22^2+22^3 = 11132$ pairs and triples. In practice only a small proportion of tuples are actually realised - see Tables 3 and 4 . At other stages of the parsing process larger tagsets are required (see [21]).

## 5.4 Rule based pruning

Strings can potentially be generated with the hypertags in all possible positions, in all possible sequences of ambiguous tags. However, this process would produce an unmanageable amount of data, so it is pruned by rule based methods integrated into the generation process. Applying local and semi-local constraints the generation of any string is zapped if a prohibited feature is produced. For fuller details see [21] or [1]. An example of a local prohibition is that the adjacent pair **(verb, verb)** is not allowed. Of course **(auxiliary verb, verb)** is permissible, as is **(verb, ] , verb)**. There is a relationship between the connectivity pattern of the Hodyne net and the probability of an adjacent tuple being a candidate for prohibition, which is explained below, Section 9.4. An example of a semi-local constraint is "if a relative pronoun or subordinating conjunction occurs in the pre-subject or in the subject, it must be followed by a verb before the closure of that constituent". These rules are similar to those in a constraint grammar [35], but are not expected to be comprehensive. There are also arbitrary length restrictions on the sentence constituents: currently, the maximum length of the pre-subject is 15 words, of the subject is 12 words.

## 5.5 Neural processing

This pruning operation is powerful and effective, but it still leaves a set of candidate strings for each sentence - typically between 1 and 25 for the technical manuals. Around 25% of sentences are left with a single string, but the rest can only be parsed using the neural selector. This averages at about 3 for the technical manuals, more for sentences from other domains.

In training we manually identify the string with the correct placement of hypertags, and the correctly disambiguated part-of-speech tags. In testing mode, the correct string is selected automatically.

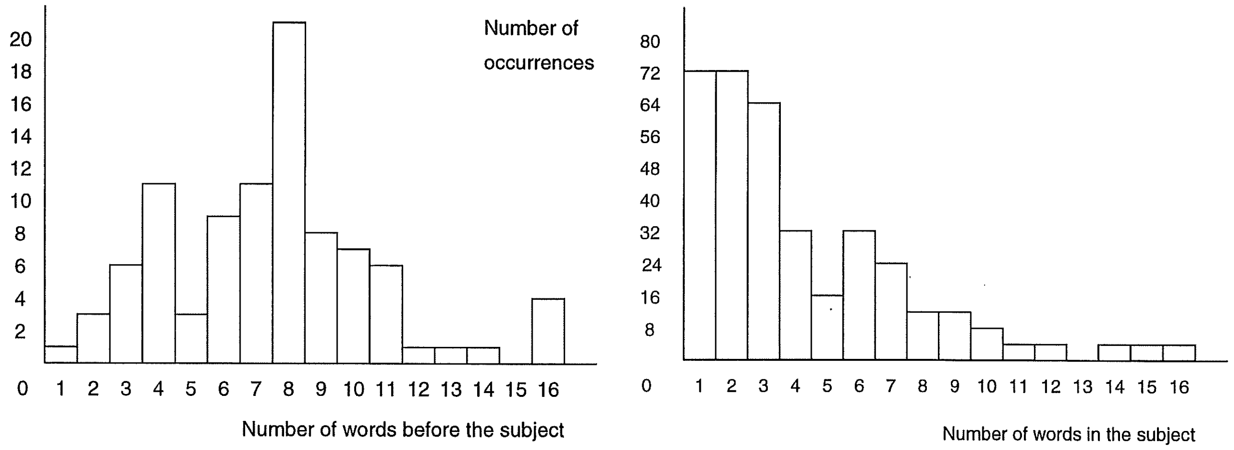

Figure 2: The frequency of constituent length for pre-subject and subject

## 6 Characteristics of the training and testing data

The domain for which our parser was developed was text from technical manuals from Perkins Engines Ltd. They were written with the explicit aim of being clear and straight forward [36]. Using this text as a basis we augmented it slightly to develop the prototype on which users try their own text.

Declarative sentences were taken unaltered from the manuals for processing: imperative sentences, titles, captions for figures were omitted. 2% of declarative sentences were omitted, as they fell outside the current bounds (e.g. the subject had more than 12 words). A corpus of 351 sentences was produced. Some statistics are given in Figure 2 and Table 1.

| | |
|---|---|
| Number of sentences | 351 |
| Average length | 17.98 words |
| No. of subordinate clauses: | |
| In pre-subject | 65 |
| In subject | 19 |
| In predicate | 136 |
| Co-ordinated clauses | 50 |

Table 1: Corpus statistics. Punctuation marks are counted as words, formulae as 1 word.

This corpus (Tr-all) was divided up 4 ways (Tr 1 to Tr 4) so that nets could be trained on part of the corpus and tested on the rest, as shown in Table 2. In order to find the placement of the subject boundary markers we do not need to analyse the predicate fully, so the part of the sentence being processed is dynamically truncated 3 words beyond the end of any postulated closing hypertag. Thus the number of pairs and triples generated represent part of the sentence only.

| Training set | number of sentences | number of strings | Test set | number of sentences | number of strings | Ratio of testing/training strings |
|---|---|---|---|---|---|---|
| Tr-all | 351 | 1037 | | | | |
| Tr 1 | 309 | 852 | Ts 1 | 42 | 85 | 0.10 |
| Tr 2 | 292 | 863 | Ts 2 | 59 | 174 | 0.20 |
| Tr 3 | 288 | 843 | Ts 3 | 63 | 194 | 0.23 |
| Tr 4 | 284 | 825 | Ts 4 | 67 | 212 | 0.26 |

Table 2: Description of training and test sets of data

| Training set | number of pairs in 'yes' strings | number of pairs in 'no' strings | Test set | number of new pairs in 'yes' strings | number of new pairs in 'no' strings |
|---|---|---|---|---|---|
| Tr-all | 162 | 213 | | | |
| Tr 1 | 161 | 211 | Ts 1 | 1 (1%) | 2 (1%) |
| Tr 2 | 160 | 210 | Ts 2 | 2 (1%) | 3 (1%) |
| Tr 3 | 156 | 210 | Ts 3 | 6 (4%) | 3 (1%) |
| Tr 4 | 149 | 193 | Ts 4 | 13 (9%) | 20 (10%) |

Table 3: Number of pairs in training and testing sets. 'Yes' indicates correct strings, 'no' incorrect ones

| Training set | number of triples in 'yes' strings | number of triples in 'no' strings | Test set | number of new triples in 'yes' strings | number of new triples in 'no' strings |
|---|---|---|---|---|---|
| Tr-all | 406 | 727 | | | |
| Tr 1 | 400 | 713 | Ts 1 | 6 (2%) | 14 (2%) |
| Tr 2 | 383 | 686 | Ts 2 | 23 (6%) | 41 (6%) |
| Tr 3 | 361 | 642 | Ts 3 | 45 (12%) | 85 (13%) |
| Tr 4 | 364 | 632 | Ts 4 | 42 (12%) | 95 (15%) |

Table 4: Number of triples in training and testing data sets

| | |
|---|---|
| Total number of pairs in 'yes' strings | 4142 |
| Total number of pairs in 'no' strings | 8652 |
| Total number of triples in 'yes' strings | 3736 |
| Total number of triples in 'no' strings | 8021 |

Table 5: Total number of tuples in Tr-all, including repetitions

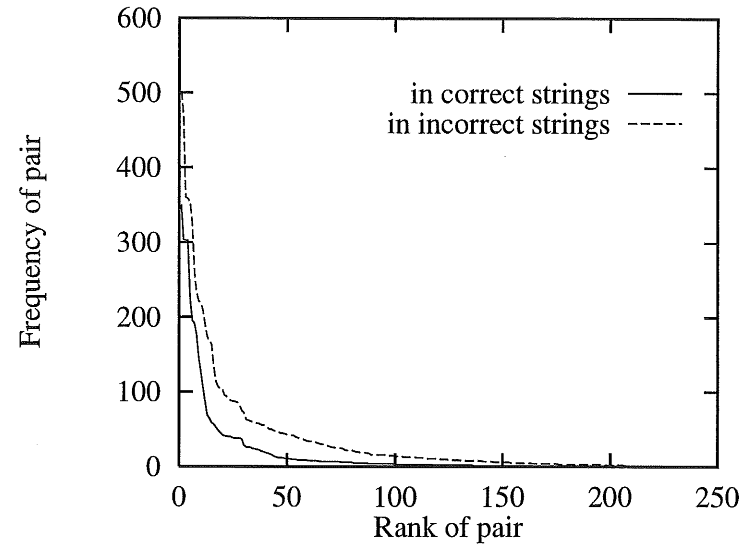

Figure 3: Data from 351 sentences in technical manuals. Pairs are ranked by frequency of occurrence in correct and incorrect strings. Relationship between rank and frequency shown.

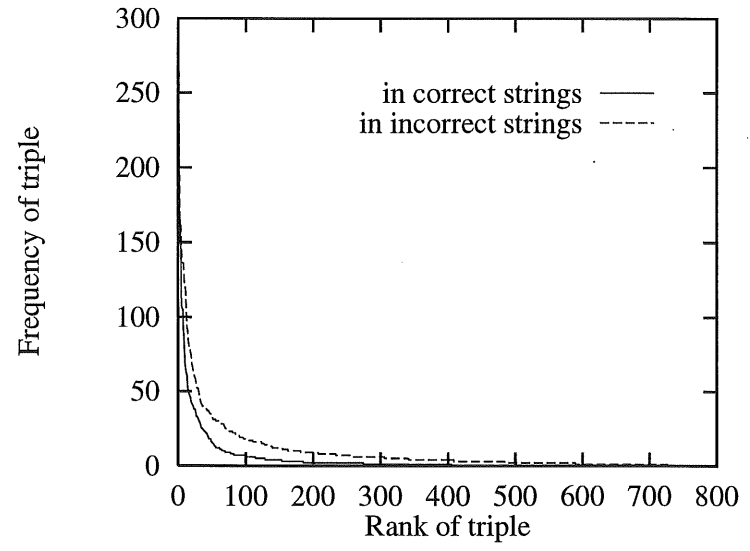

Figure 4: Relationship between rank and frequency of triples on the same data.

We find that the distribution of data has a "zipfian" character, as Figures 3 and 4 show. Consequently, a significant number of tuples occur in the test set, but have not occurred in the training set (Tables 3 and 4). Though this could be put down to the small size of the data sets, it is in fact typical of much larger corpora. See, for instance, an analysis of tag combinations in the 1 million word LOB corpus [37]. To illustrate this point consider the following unremarkable sentence:

The directions given below must be carefully followed. ... (4)

which should return

[ The [ directions ] given below ] must be carefully followed. ... (4a)

Now, this is not a particularly unusual construction, but it is quite rare all the same. In the LOB corpus the pair **(preposition, modal-verb)**, which represents the words **(below, must)** has a frequency of less than 0.01%, if it occurs at all [37]. A significant proportion of data is accounted for by rare examples.

## 6.1 Interpreting input and output

The input and output modules are the same for all the nets we use. For training, the set of strings generated by the training text is taken as a whole. Results are interpreted differently in testing mode: here we consider separately the set of strings generated for each sentence. Each string is given a "grammaticality measure " $\Gamma$, and the string with the highest $\Gamma$ measure is taken as the correct one for that sentence. In training, a correct string is required to have a positive $\Gamma$ measure, an incorrect string a negative one. There are 4 metrics for correctness. The measure "correct-a" requires that the hypertags are correctly placed, "correct-b" requires also that all words within the subject are correctly tagged, "correct-c" that all words within the part of the sentence being processed are correctly tagged. The final measure "correct-d" records the proportion of strings that are in the right class. It can happen that the highest scoring string may have a negative $\Gamma$, but will be selected as the least bad even though it is in the wrong class. Conversly, some incorrect strings can have a positive $\Gamma$ without having the highest score.

When a sentence like (4) above is processed this particular construction will probably not have occurred in a training string. However, when we generate candidate strings wrong placements should be associated with stronger negative weights *somewhere* in the string. For example, in the structure

* [ The directions given ] below must be carefully followed. ... (4b)

the proposed subject would not be associated with strong negative weights, but the following pairs and triples include at least one that is strongly negative, such as **(preposition, auxiliary verb)**. [4] This would have been an element in one of the negative strings generated in the training set. The correct placement, as in (4a) would be the least bad, the one with the highest $\Gamma$ score.

For practical purposes the measures "correct-a", "-b", and "-c" will be the significant ones. But in analysing the performance of the networks we will be interested in "correct-d", the extent to which the net can generalize and correctly classify strings generated by the test data.

[4]The tag class "auxiliary verb" includes modal verbs in our system.

### Metrics of other systems

Note that these measures relate to a string, not to individual elements of the string. This contrasts with some natural language processing systems, in which the measure of correctness relates to each word. For instance, automated word tagging systems typically measure success by the proportion of *words* correctly tagged. The best stochastic taggers typically quote success rates of 95% to 97% correct. If sentences are very approximately 20 words long on average, this can mean that there is an error in many sentences.

## 7 Using a single layer net

For a feed forward network operating in supervised mode the simplest processor is a single layer network. If the data is not linearly separable, as must be assumed in the general case, then of course a multi-layer network is necessary. However, it is good practice to start by investigating the use of single layer networks. If data is not linearly separable it may be possible to transform it to a separable representation, so that single layer nets are appropriate processors.

### 7.1 Conversion to linearly separable forms

It is always theoretically possible to solve supervised learning problems with a single layer, feed forward network, providing the input data is enhanced in an appropriate way. A good explanation is given by Pao [38]. Whether this is desirable in any particular case must be investigated. The enhancement, in general a functional transform, can map the input data onto a space, usually of higher dimensionality, where it will be linearly separable. Widrow's valuable 1990 paper on "Perceptron, Madaline, and Back Propagation" [39, page 1420] explores these approaches "which offer great simplicity and beauty".

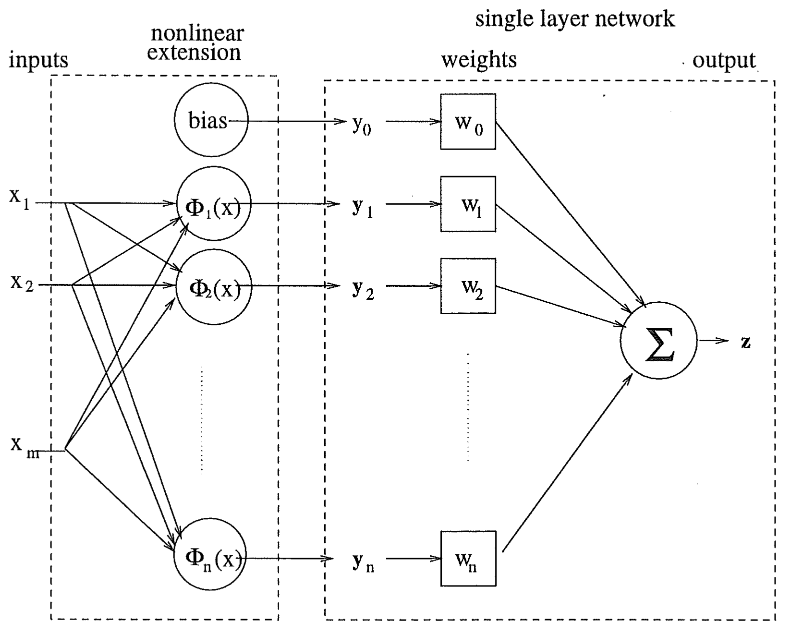

Figure 5: The Generalized Single Layer Network, GSLN, with 1 output

Figure 5 illustrates the form of the Generalized Single Layer Network (GSLN). This figure is derived from Holden and Rayner [3]. The inputs $\{x\}$ are preprocessed, and converted to elements $\{y\}$. The single layer net will then process these elements to produce an output $\mathbf{z}$, temporarily assuming 1 output. The preprocessing functions, $\Phi$, also called the basis functions, can take various forms. They can be applied to each input node separately, or, as illustrated, they can model the higher order effects of correlation. In our processor $\Phi$ is an ordered 'AND'. This function is also used in the grammatical inference work of Giles et al. [30, 40]. The $\Phi$ function can also be arithmetic: for instance in *polynomial networks* the elements of the input vectors are combined as products [13, page 8]. This approach has been successfully used for the automated interpretation of telephone company data in tabular form [41].

Radial basis function (RBF) networks also come into the class of GSLNs. In their two stage training procedure the parameters governing the basis functions are determined first. Then what is effectively a single layer net is used for the second stage of processing. Examples include Tarassenko's work on the analysis of ECG signals [42].

An important characteristic of the GSLN is that processing at different layers is de-coupled. The first stage of training is unsupervised: the $\Phi$ functions are applied without recourse to desired results. In the second stage of training a supervised method is used, in which weight adjustments on links are related to target outputs.

One perspective on the GSLN is given by Bishop [12, e.g. page 89], who characterises this type of system as a special case of the more general multilayer network. Whereas in the general case the basis functions at the first layer are modified during the training process, in this type of system the basis functions are fixed independently. Widrow [39, page 1420] puts it the other way round: "one can view multilayer networks as single layer networks with trainable preprocessors...".

## 7.2 The conversion function used in the parser

Minsky and Papert [11] and Pao [38] investigated the conversion of input data items to higher order elements: in our case the higher order elements also preserve sequential order. For a sequence that includes the elements

$$\ldots x_i x_{i+1} x_{i+2} \ldots$$

the conversion function $\Phi$ for the case of trigrams is given by

$$\Phi_i(x) = (x_i, x_{i+1}, x_{i+2})$$

which represents the ordered 'AND'.

This function was derived using heuristic methods, but the approach was supported by an objective analysis of the proposed representation. We aimed to capture *some* of the implicit information in the data, model invariances, represent structure. As described in Section 4.3, the choice of the tupling pre-processing function is supported by information theoretic analysis. It captures local, though not distant, dependencies. Using this representation we address simultaneously the issues of converting data to a linearly separable form, modelling its sequential character and capturing some of its structure.

Taking data items as pairs typically produces training sets of which about 97% can be learnt by a single layer network; taking triples raises learnability above 99% (see Table 10). Thus our data is almost linearly separable. On test data results above 90% are typically returned (see Tables 6, 7 and 8).

### 7.3 The practical approach

Minsky and Papert acknowledged that single layer networks could classify linearly inseparable data if it was transformed to a sufficiently high order [11, page 56], but claimed this would be impractical. They illustrated the point with the example of a parity tester. However, this example is the extreme case, where *any* change of a single input element will lead to a change in the output class. If there are $n$ inputs, then it is necessary to tuple each element together to $O(n)$. The consequent explosion of input data would make the method unusable for all but the smallest data sets. Indeed, neural methods in general are not appropriate for the parity detection task.

However, in practice, real world data may be different. Shavlik et al [43] compare single and multi-layer nets on 6 well known problems, and conclude "Regardless of the reason, data for many 'real' problems seems to consist of linearly separable categories..... Using a Perceptron as an initial test system is probably a good idea." Herman and Yeung (1992) [44] discuss the extent to which the simple Perceptron works in real world classification tasks. They recommend that for a new problem classification by a linear separator should be the method of choice until the evidence indicates that this is inadequate.

This empirical approach is advocated here. Tests for linear separability and related problems are computationally heavy [45, 46], so we tried single layer networks to see whether the higher order data we use is in practice linearly separable, or nearly so.

### 7.4 Training set size

There is a relationship between training set size and linear separability. Cover's classical work addressed the probability that a set of *random*, real valued vectors with random binary desired responses are linearly separable [47, 39]. Using his terminology and taking the term "pattern" to mean a training example, the critical factor is the ratio of number of patterns, $\Pi$, to number of elements in each pattern, $n$. While $\Pi/n < 1$, the probability $P_{separable} = 1.0$. If $\Pi/n = 1$ then $P_{separable} = 0.5$. As $\Pi/n$ increases, $\Pi_{separable}$ quickly declines.

These observations are given as background information to indicate that training set size should be considered, but they do not apply in our case as they stand. First, our data is not random. Secondly, a necessary condition that the vectors are in "general position", normally satisfied by real valued vectors, may not hold for binary vectors [48, page 97].

Recall (Section 2.3) that the number of training examples, $\Pi$, is a factor in determining generalization capability. Decreasing $\Pi$ to convert data to a linearly separable form would be profitless.

The ratio of training examples to weights in our data is shown in Figure 6. Note that the corpus used for this preliminary working prototype is small compared to other corpora, and future work will use much larger ones, which could affect this ratio.

## 8 Three single layer networks

### 8.1 Architecture

Refer again to Figure 5, illustrating the GSLN. In this work we compare 3 networks, but all use the same pre-processing $\Phi$ function, described in Section 7.1. We now compare methods of processing at the second stage, that is the performance of 3 different single layer classifiers. The Perceptron and LMS net can be characterised as examples of the GSLN in Figure 5. The Hodyne model differs in having 2 outputs, not symmetrically connected, as in Figure 7.

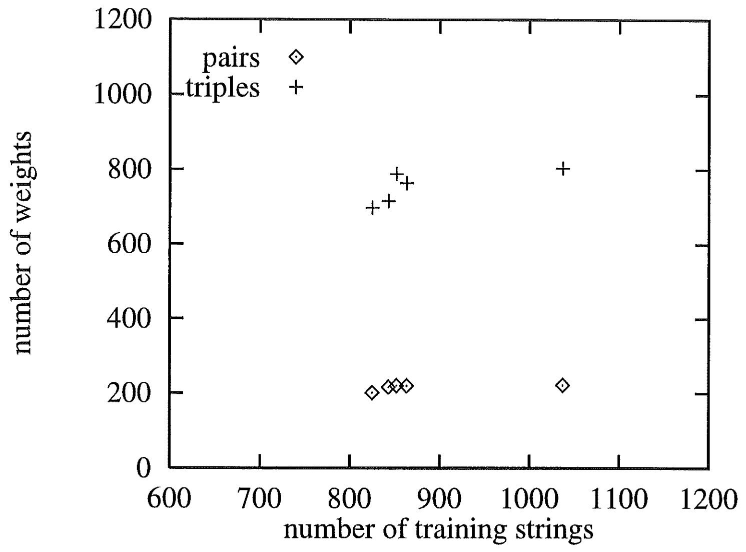

Figure 6: Relationship between number of training examples and number of weights, for the Perceptron and LMS nets with one output.

## 8.2 Methods of adjusting connection weights during training

When single layer networks are used, we do not have the classic problem of "credit assignment" associated with multi-layer networks: the input neurons responsible for incorrect output can be identified, and the weights on their links adjusted. There is a choice of methods for updating weights. We do not *have* to use differentiable activation functions, as in the multi-layer Perceptron. These methods can therefore be divided into two broad categories. First there are "direct update" methods, secondly, there are "error minimization" approaches, which can also be used in multi-layer nets. In the first method, used in the traditional Perceptron and Hodyne-type nets, a weight update is invoked if a training vector falls into the wrong class. This approach is related to the ideas behind reinforcement learning, but there is no positive reinforcement. If the classification is correct weights are left alone. If the classification is incorrect then the weights are incremented or decremented. No error measure is needed: the weight update is a function either of the input vector (Perceptron), or of the existing weights (Hodyne).

In the second method of weight adjustment an error measure is used, based on the difference between a target value and the actual value of the output node. This is frequently, as in standard back propagation, a process based on minimizing the mean square error, to reach the LMS error [49].

Now, the traditional LMS measure can lead to situations where most input vectors are close to the target, while a few, or a single one, are distant. This may be desirable when noisy data is processed, but not for our linguistic data, where we want precise fitting. We want to capture information from isolated training examples, and to classify strings that are ungrammatical in a single element as well as those that are grossly ungrammatical. Therefore, for this experiment we have compared different forms of "direct update" to an error minimization technique. The algorithms for the three networks are given in the Appendix.

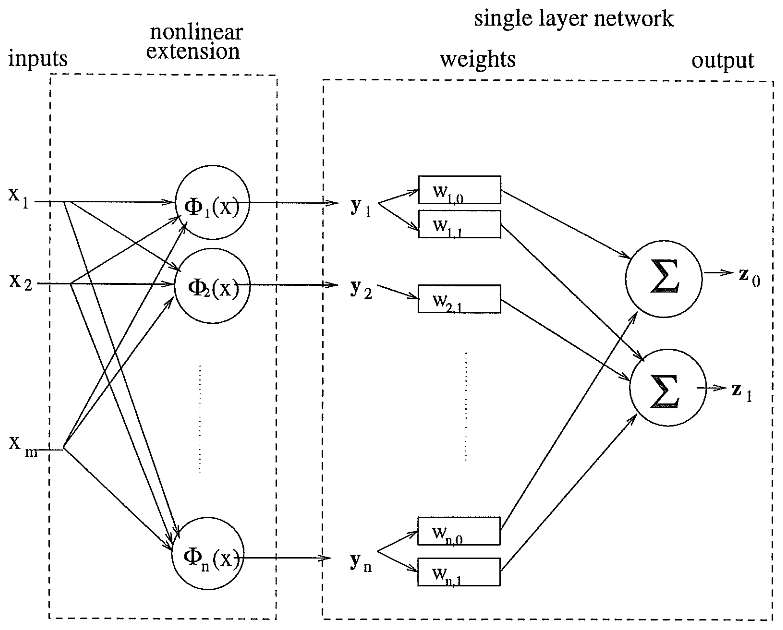

Figure 7: The Hodyne network. Note that it is not fully connected during training

## 9 Performance

### 9.1 Training

Since we have been developing a prototype the training threshold was taken so that training was fast - less than 10 seconds for the Perceptron, less than 20 seconds for the Hodyne network. Subject to this constraint the percentage of strings that could be trained ranged from 96.5% to 99.0%. See Table 10 in the Appendix. The Perceptron was fastest. For the LMS net training times were between 1 and 2 orders of magnitude greater. However, note that the traditional method of gradient descent employed here has long been known to be inefficient [50, 12]. Error minimization techniques should normally use other methods, such as conjugate gradients.

### 9.2 Ability to generalize

Results are given in Tables 6 to 8. This system has been developed to produce a winning string for each sentence, and performance can be assessed on different measures of correctness, as described in Section 6.1. For the purpose of investigating the function of the networks we take the strictest measure, *correct-d* in the tables, requiring that strings should be classified correctly. However, we can interpret the results so that in practice we get up to 100% correct for our practical application. To do this we take the "winning string" for each sentence. Now, the one we want may be in the wrong class, classified as ungrammatical, but if it is the "least bad" string it can still be the winner. In a similar way, a string we do not want selected may be wrongly classified as grammatical, but if it is not the "best" string it will not be chosen. Thus the practical measure of correctness can be higher than the percentage of correctly classified strings.

| Training set | Test set | Pairs used | Triples used | Training threshold | correct-a % | correct-b % | correct-c % | correct-d % |
|---|---|---|---|---|---|---|---|---|
| Tr 1 | Ts 1 | Y | Y | 95.0 | 100 | 97.6 | 95.2 | 92.9 |
| | | Y | Y | 99.0 | 100 | 97.6 | 95.2 | 89.4 |
| | | Y | | 97.0 | 100 | 97.6 | 92.9 | 85.8 |
| | | | Y | 98.5 | 100 | 97.6 | 97.6 | 91.8 |
| Tr 2 | Ts 2 | Y | Y | 99.0 | 96.6 | 96.6 | 91.5 | 88.5 |
| | | Y | | 98.0 | 93.2 | 93.2 | 88.1 | 89.1 |
| | | | Y | 98.5 | 98.3 | 98.3 | 89.8 | 85.6 |
| Tr 3 | Ts 3 | Y | Y | 99.0 | 98.4 | 98.4 | 95.2 | 80.9 |
| | | Y | | 96.0 | 96.8 | 93.7 | 92.1 | 85.6 |
| | | | Y | 99.0 | 96.8 | 95.2 | 90.5 | 78.9 |
| Tr 4 | Ts 4 | Y | Y | 99.0 | 92.5 | 92.5 | 74.6 | 77.4 |
| | | Y | | 97.0 | 92.5 | 92.5 | 92.5 | 82.5 |
| | | | Y | 99.0 | 89.6 | 88.1 | 83.6 | 78.3 |

Table 6: Results using Perceptron. Recall that correct-a means hypertags are correctly placed, correct-b that words inside subject are correctly tagged also, correct-c that all words in part of sentence being processed are correctly tagged also, correct-d that the string is in the right class

Table 9 gives a summary of the results, showing how these vary with the ratio of test set size to training set size. This would be expected. If this ratio is not sufficiently small performance degrades sharply.

The Hodyne net performed well, and this architecture was used for the prototype.

The slightly inferior results for the Perceptron could be related to methods of initialization. The random initial weights allocated to links in the Perceptron contrast with the set level in the Hodyne net. Though this set level is arbitrary, the figure is chosen to accord with our linguistic intuition. We observe that a significant number of training examples contain infrequent events, and we wish these events to play their due part in the classification process. Thus, they are initialized to a significant value. Now, the total training data can be separated into its two classes in many ways: the separating surface is not uniquely specified. Therefore, the level at which weights end up when the training threshold is passed may still bear some relation to their inititial values.

This hypothesis accords with the results of some earlier work carried out in the field, when the performance of a multi layer Perceptron was compared to single layer models [21]. The MLP performed best when "guided initialization" was used. With this method the initial random weight settings were pushed above or below a threshold in accordance with an "anticipated" result. (Even with guided initialization, however, the MLP performed less well than the single layer network.)

## 9.3 Distribution of trained weights

After training we see that the distribution of weights in Hodyne and Perceptron nets have certain characteristics in common. In both cases there is a trend for links on the least common input tuples to be more heavily weighted than the more common: see Figures 8 and 9.

This can be understood when we examine the process by which the weights are adapted. Since

| Training set | Test set | Pairs used | Triples used | Training threshold | correct-a % | correct-b % | correct-c % | correct-d % |
|---|---|---|---|---|---|---|---|---|
| Tr 1 | Ts 1 | Y | Y | 95.0 | 100 | 100 | 97.6 | 91.8 |
| | | Y | Y | 99.0 | 100 | 100 | 100 | 92.9 |
| | | Y | | 97.0 | 100 | 97.6 | 95.2 | 90.6 |
| | | | Y | 98.5 | 100 | 100 | 100 | 89.4 |
| Tr 2 | Ts 2 | Y | Y | 99.0 | 100 | 100 | 94.9 | 91.4 |
| | | Y | | 98.0 | 100 | 100 | 94.9 | 90.2 |
| | | | Y | 98.5 | 98.3 | 98.3 | 94.9 | 96.8 |
| Tr 3 | Ts 3 | Y | Y | 99.0 | 100 | 98.4 | 95.2 | 89.2 |
| | | Y | | 96.0 | 98.4 | 98.4 | 95.2 | 91.2 |
| | | | Y | 99.0 | 96.8 | 96.8 | 92.1 | 86.1 |
| Tr 4 | Ts 4 | Y | Y | 99.0 | 95.5 | 94.0 | 91.0 | 84.4 |
| | | Y | | 98.0 | 95.5 | 94.0 | 92.5 | 83.0 |
| | | | Y | 99.0 | 94.0 | 94.0 | 83.6 | 82.5 |

Table 7: Results for Hodyne net on same training and testing data as for the Perceptron (Table 6)

| Training set | Test set | Pairs used | Triples used | Training threshold | correct-c % | correct-d % |
|---|---|---|---|---|---|---|
| Tr 1 | Ts 1 | Y | Y | 99.0 | 97.6 | 92.9 |
| | | Y | | 97.0 | 90.5 | 88.3 |
| | | | Y | 99.0 | 97.6 | 90.6 |
| Tr 2 | Ts 2 | Y | Y | 99.0 | 91.5 | 88.5 |
| | | Y | | 98.0 | 88.1 | 88.5 |
| | | | Y | 98.5 | 93.2 | 85.1 |
| Tr 3 | Ts 3 | Y | Y | 99.0 | 93.7 | 85.1 |
| | | Y | | 96.0 | 95.2 | 87.6 |
| | | | Y | 99.0 | 90.5 | 80.9 |
| Tr 4 | Ts 4 | Y | Y | 99.0 | 92.5 | 82.5 |
| | | Y | | 97.0 | 97.0 | 83.5 |
| | | | Y | 99.0 | 85.1 | 80.2 |

Table 8: Results for LMS net on the same data. Compare with Tables 6 and 7

| Ratio test set / training set | Perceptron % test strings correct | Hodyne % test strings correct | LMS % test strings correct | Hodyne % hypertags correct |
|---|---|---|---|---|
| 0.10 | 89.4 | 92.9 | 92.9 | 100 |
| 0.20 | 88.5 | 91.4 | 88.5 | 100 |
| 0.23 | 80.9 | 89.2 | 85.1 | 100 |
| 0.26 | 77.4 | 84.4 | 82.5 | 95.5 |

Table 9: Summary of results culled from Tables 2, 6, 7 and 8, showing performance on 4 different training and test sets.

we are processing negative as well as positive examples in the training stage, the movement of weights differs from that found with positive probabilities alone. A significant number of nodes represent those tuples that have never occurred in a grammatical strings. A few nodes represent tuples that have only occurred in grammatical strings (Tables 3 and 4). However, some very common tuples will appear frequently in both correct and incorrect strings. Consider a pair such as **(start-of-sentence, open-subject)**. This will often occur at the start of both grammatical and ungrammatical strings. The result of the learning process is to push down the weights on the links to both the "yes" and the "no" output nodes.

This effect accompanies the decision to assess the correctness of a whole string, rather than the status of each element. Strings that are slightly wrong will include tuples that occur both in grammatical and ungrammatical sequences. As a consequence we see that the classifcation decision can depend more on infrequently occurring tuples. In particular, tuples that usually only occur in an ungrammatical string can have a significant influence on the classification task.

### 9.4 Hodyne's pattern of connectivity and the Prohibition Table

Any single entry in the preliminary Prohibition Table can be omitted, and the pair or triple be included in the neural processing task. In this case a tuple that cannot occur in a grammatical string will, for Hodyne, only be connected to the "no" output node. Conversly, if we examine the linkage of the Hodyne net, those tuples that are only connected to the "no" output are candidates for inclusion in a constraint based rule. Of course, with zipfian distribution, there is a chance that a rare grammatical occurrence may show up as the size of the training set increases, so these tuples are only candidates.

## 10 Conclusion

We conclude that the approach described in this paper is appropriate for natural language processing. Linguistic data is a suitable candidate for conversion to a linearly separable form, so that it can subsequently be processed by a single layer network.

The analysis in the previous section illustrates the transparency of single layer networks, and indicates why they are such convenient tools. The Hodyne net in particular lends itself to further linguistic analysis. The speed of training, measured in seconds, shows how fast single layer networks can fix their weights. Training times are hardly an issue. A significant question of generalization ability is seen to relate to the ratio of testing to training data set size. It is important that this ratio is kept small, and our work will be continued with much larger corpora.

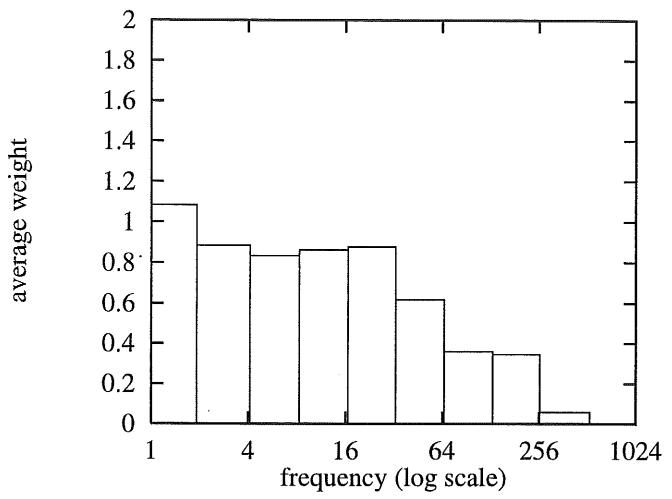

Figure 8: Weights plotted against frequency of input node occurring. Hodyne net, training corpus Tr 1

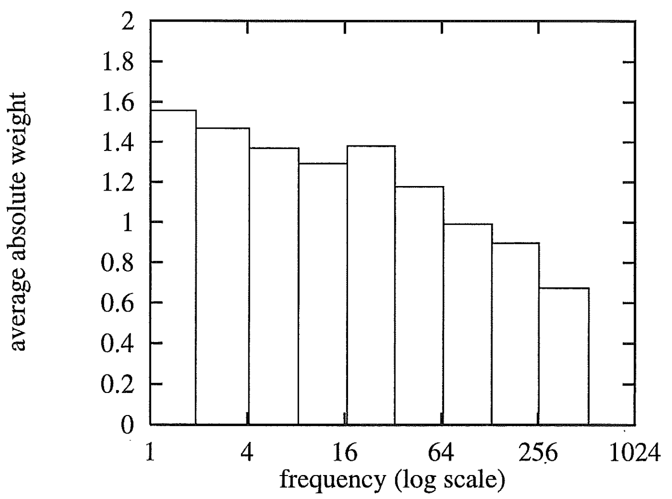

Figure 9: Weights plotted against frequency of input node occurring. Perceptron net, training corpus Tr 1

The general conclusion to this paper is that it is worth investigating ways of pre-processing data so that single layer networks can be used. Compared to multi-layer perceptrons, this approach has the advantage of fast two stage training. Furthermore, the parameters of the processor are more amenable to being interpreted. Arguments against this approach centre on the lack of a principled method to find the pre-processing, $\Phi$ function. But though the methods of finding the initial $\Phi$ function are based on intuition, a close initial examination of the data means that this intuition is founded on an understanding of the data characteristics. In the case of the linguistic data this has included the use of information theoretic tools. The setting of parameters is not data driven at the micro level, as in a supervised learning environment, but it should be in the sense that the functions are chosen to capture some of the structure and invariances of the data.

# Appendix

## Algorithms for training the 3 networks

### Notation and its interpretation

Let each input vector have $n$ elements $y_i$. Let $w(t)_{ij}$ be the weight from the $i$th input node to the $j$th output node at time $t$. For the Perceptron and the LMS nets with one output this reduces to a vector, so that $w(t)_i$ is the weight from the $i$th node to the output at time $t$. The output from the GSLN is represented by $z$. For the Hodyne net, the 2 outputs are $z_0$ and $z_1$, where $z_0$ represents the "yes, correct" output and $z_1$ represents the "no, incorrect" output.
The following notation is used

$u(t)_{ij}$ the update factor added to weight $w(t)_{ij}$

$\eta$ the learning rate which scales $u$,

$\mu$ the momentum term, used in the LMS algorithm

$f(.)$ a non-linear activation function; the bipolar sigmoid is used in the LMS algorithm

$d$ the desired output.

$\tau$ the tolerance, used if the activation function is asymptotic to its bounds.

$\delta = -1$ or $\delta = +1$ indicates whether weights should be decremented or incremented Recall $\Gamma$ is the grammaticality measure. For the GSLN nets $\Gamma = z$, which is bounded by $+1$ and $-1$. For Hodyne $\Gamma = z_0 - z_1$. In testing mode the strings generated by each sentence are processed, and the string with the highest $\Gamma$ score for that sentence is the winner.

## I The Perceptron training algorithm

A brief outline of the way this well known algorithm has been implemented follows.

```
Mark all links disabled, apart from the bias represented by $y_0$.
Set $\eta = 0.1$.
(The Perceptron convergence proof depends on $\eta$ being sufficiently small.)
Initially, percentage of strings correctly classified = 0.0
REPEAT
  from START1 to END1 until % strings correctly classified exceeds chosen threshold:
START1
      REPEAT from START2 to END2 for each string
      START2
            Present input, a binary vector
            Normalize, producing a real vector $y_1, y_2, ....y_n$
            For any $y > 0.0$, enable link if it is disabled
            Initialize weight on any new link to random value between $-0.3$ and $0.3$
            Calculate output: $z = \sum_{i=0}^{i=n} w_i * y_i$
            Present desired output, $d > 0$ or $d <= 0$
            If actual result = desired result
                  Count string correct, leave weights alone
            Else adjust weights on current active links
                  if $d > 0$ and $z < 0$ then $\delta = 1$, else $\delta = -1$
                  $w(t+1)_i = w(t)_i + \delta * \eta * y_i$
      END2
      Calculate % strings correctly classified. If greater than threshold, terminate
END1
```

When training is completed all links are enabled, and weights on new links are set to 0.0 for testing. Tuples in test data that did not occur in the training data contribute nothing to the classification decision.

## II Hodyne training algorithm and pattern of connectivity

This network is derived from the model introduced by Wyard and Nightingale [4]. More implementation details can also be found in [21]. A summary of the method follows.

```
Mark all links disabled.
Initially, percentage of strings correctly classified = 0.0
REPEAT
  from START1 to END1 until % strings correctly classified exceeds chosen threshold:
START1
      REPEAT from START2 to END2 for each string
      START2
            Present input, a binary vector, $y_1, y_2, ....y_n$
            Present desired output, $z_0 > z_1$ or vice versa
            For any $y > 0.0$ enable link to desired result if it is disabled
            Initialize weight on any new link to 1.0
            Calculate outputs $z_0$ and $z_1$
                      $z_k = \sum_{i=1}^{i=n} w_{i,k} * y_i$
            If actual result = desired result
                      Count string correct, leave weights alone
            Else adjust weights on current active links:
                      if $z_0 < z_1$ then $\delta = +1$ on links to $z_0$, $\delta = -1$ on links to $z_1$
                      and vice versa

                 $$w(t+1)_{i,j} = \left[1 + \frac{\delta * w(t)_{i,j}}{1 + (\delta * w(t)_{i,j})^4}\right] w(t)_{i,j}$$

      END2
      Calculate % strings correctly classified. If greater than threshold, terminate
END1
```

Note that during training elements of a new input vector may be linked to both, either or neither output node. This represents the fact that a tuple can appear (i) in both a correct and incorrect string, (ii) in either or (iii) in neither. Tables 3 and 4 gives some information on the distribution of elements in training and testing sets. When training is completed weights on all disabled links are enabled and set to 0.0 for testing, as for the reasons given above for the Perceptron.

For the Hodyne type net the update factor $u$ is a function of the current weights; as the weights increase it is asymptotic to 0, as they decrease it becomes equal to 0.

$$u(t)_{i,j} = \frac{\pm w(t)_{i,j}^2}{1 \pm w(t)_{i,j}^4}$$

This function satisfies the requirement that the weights increase monotonically and saturate (see Figure 10). We use the original Hodyne function with a comparatively low computational load. Note that in contrast to the Perceptron, where the learning rate is set at compile time, the effective learning rate in this method varies dynamically.

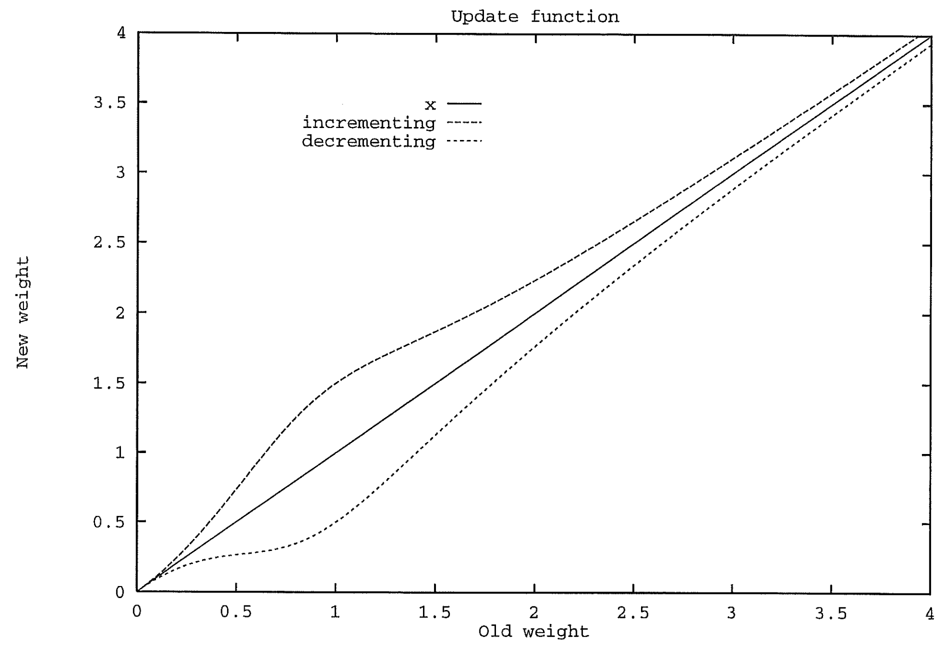

Figure 10: Relationship between old and new weights for the Hodyne net

## III Training algorithm for the LMS network

This is based on the traditional method, described in "Parallel Distributed Processing" [49, page 322]. It has been known for many years in the numerical optimization field that this gradient descent technique is a poor, slow method [50]. This is now also accepted wisdom in the neural network community [12]. Other training methods, such as conjugate gradients, are usually preferable. For this experiment the traditional method has been used, but some variations to speed up training and improve performance have been incorporated. Brady et al. [51] described some anomalies that can arise with the traditional LMS model. As a remedy Sontag's technique for interpreting the error measure is included [52]. This means that an error is only recorded if a vector falls into the wrong class. The target output is a threshold, and if this threshold is passed the vector is considered correctly classified. This contrasts with the original LMS method, in which an error is recorded if the target is either undershot or overshot.

With this network the activation of the single output node is bounded by +1 and −1, as for the Perceptron. The algorithm is based on finding a minmum value for the error between desired and actual outputs, summed over the whole training set.

The process of "Guided Initialization" sets initial random values within bounds determined by the expected output. To implement this in training, see whether a previously unseen input element belongs to "yes" or "no" string, corresponding to desired outputs of +1 or −1. Then set random value between 0.0 and 0.3 for "yes", between 0.0 and −0.3 for "no". [21]

### Algorithm

```
Mark all links disabled.
Set learning coefficient $\eta = 0.6$, momentum $\mu = 0.4$, tolerance $\tau = 1.0$
These parameters are found by trial and error.
Initially, % strings correctly classified = 0.0
REPEAT
  from START1 to END1 until % strings correctly classified exceeds chosen threshold:
START1
      REPEAT from START2 to END2 for each string
      START2
            Present input, a binary vector
            Normalize, producing a real vector $y_1, y_2, ....y_n$
            For any $y > 0.0$, enable link if it is disabled
            Initialize weight on any link enabled this time, with "Guided Initialization"
            Calculate output, using bipolar sigmoid
                  $f(x_i) = \frac{2.0}{1.0+\exp^{-w_i * y_i}}$
                  $z = \sum_{i=0}^{i=n} f(x_i)$
            Calculate error $d - z$ for this string
            If absolute value of error $< \tau$
                  Count string correct, leave weights alone
            Else calculate the update factor $u_i$ for current active links, and store in a cumulative array
                  $u_i = \eta * (d - z)$
      END2 (Now all strings have been processed in this cycle)
      Calculate % strings correctly classified. If greater than threshold, terminate
      Nrmalize update factors for each weight, $u(t)_i$
      Save for next momentum term
      Adjust weights
            $w(t+1)_i = w(t)_i + u(t)_i + \mu * u(t-1)_i$
END1
```

| Training set | Test set | Pairs used | Triples used | Training threshold % | Training time in secs. Hodyne | Training time in secs. Perceptron | Training time in secs. lms net |
|---|---|---|---|---|---|---|---|
| Tr 1 | Tr 1 | Y | Y | 99.0 | 8 | 6 | 90 |
| | | Y | | 97.0 | 12 | 3 | 161 |
| | | | Y | 98.5 | 2 | 3 | 62 |
| Tr 2 | Tr 2 | Y | Y | 99.0 | 3 | 8 | 103 |
| | | Y | | 98.0 | 11 | 6 | 182 |
| | | | Y | 98.5 | 3 | 3 | 69 |
| Tr 3 | Tr 3 | Y | Y | 99.0 | 2 | 4 | 79 |
| | | Y | | 96.0 | 3 | 2 | 82 |
| | | | Y | 99.0 | 2 | 3 | 58 |
| Tr 4 | Tr 4 | Y | Y | 99.0 | 2 | 5 | 77 |
| | | Y | | 98.0 | 13 | 4 | 125 |
| | | | Y | 99.0 | 1 | 3 | 49 |
| Tr-all | Tr-all | Y | Y | 99.0 | 5 | 6 | 135 |
| | | Y | | 96.5 | 10 | 8 | 152 |
| | | | Y | 98.5 | 2 | 3 | 69 |

Table 10: Training times: the training threshold is set at the highest level at which Hodyne training can be completed within 20 seconds.

## References

[1] C Lyon and R Dickerson. A fast partial parse of natural language sentences using a connectionist method. In *7th Conference of the European Chapter of the Association of Computational Linguistics*, 1995.

[2] C Lyon and R Frank. Neural network design for a natural language parser. In *International Conference on Artificial Neural Networks (ICANN)*, 1995.

[3] S Holden and P Rayner. Generalization and PAC learning: Some new results for the class of generalized single layer networks. *IEEE Trans. on Neural Networks*, 1995. ( PAC is Probably Approximately Correct).

[4] P J Wyard and C Nightingale. A single layer higher order neural net and its application to context free grammar recognition. *Connection Science*, 4, 1990.

[5] H Cunningham, R J Gaizauskas, and Y Wilks. A general architecture for text engineering (GATE). Technical report, Institute for Language, Speech and Hearing (ILASH), University of Sheffield, 1996.

[6] E Black, R Garside, and G Leech. *Statistically Driven Computer Grammars of English: the IBM/Lancaster approach.* Rodopi, 1993.

[7] E M Gold. Language identification in the limit. *Information and Control*, 10, 1967.

[8] D Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45, 1980.

[9] E Charniak. *Statistical Language Learning.* MIT Press, 1993.

[10] E Brill et al. Deducing linguistic structure from the statistics of large corpora. In *DARPA Speech and Natural Language Workshop*, 1990.

[11] M Minsky and S Papert. *Perceptrons.* MIT Press, 3rd edition, 1988. Epilog added to 1969 edition.

[12] C M Bishop. *Neural Networks for Pattern Recognition.* OUP, 1995.

[13] S Holden and M Niranjan. On the practical applicability of VC dimension bounds. Technical report, Cambridge University Engineering Department, 1994.

[14] C E Shannon. Prediction and Entropy of Printed English. *Bell System Technical Journal*, 1951.

[15] G K Zipf. *Human Behaviour and the Principle of Least Effort.* Addison Wesley, 1949.

[16] T Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 1993.

[17] E Atwell. Constituent-likelihood grammar. In R Garside, G Leech, and G Sampson, editors, *The Computational Analysis of English: a corpus-based approach.* Longman, 1987.

[18] J Kupiec. Robust part-of-speech tagging using a hidden Markov model. *Computer Speech and Language*, 1992.

[19] F Jelinek et al. Classifying words for improved statistical language models. In *ICASSP*, 1990.

[20] V Gupta, M Lennig, and P Mermelstein. A language model for very large vocabulaty speech recognition. *Computer speech and Language*, 1992.

[21] C Lyon. *The representation of natural language to enable neural networks to detect syntactic structures.* PhD thesis, University of Hertfordshire, 1994.

[22] R Pocock and E Atwell. Treebank trained probabilistic parsing of lattices. School of Computer Studies, Leeds University, 1994. In the Speech-Oriented Probabilistic Parser Project: Final Report to MoD.

[23] E Charniak and E Santos. A connectionist context-free parser. In *Procs. of 9th Annual Conference of Cognitive Science Society*, 1987.

[24] J L Elman. Distributed representations, simple recurrent networks and grammatical structure. *Machine Learning*, 1991.

[25] J Feldman. Structured connectionist models and language learning. *Artificial Intelligence Review*, 1993.

[26] R Miikkulainen. *Subsymbolic Natural Language Processing.* MIT Press, 1993.

[27] T Sejnowski and C Rosenberg. NETtalk: a parallel network that learns to read aloud. In J Anderson, A Pellionisz, and E Rosenfeld, editors, *Neurocomputing.* MIT Press, 1990. Written in 1986.

[28] M Nakamura, K Maruyama, T Kawabata, and K Shikano. Neural network approach to word category prediction for English texts. In *ICASSP*, 1989.

[29] J Benello, A W Mackie, and J A Anderson. Syntactic category disambiguation with neural nets. *Computer Speech and Language*, 1989.

[30] C L Giles et al. Higher order recurrent networks and grammatical inference. In D S Touretzky, editor, *Advances in neural information processing systems.* Morgan Kaufmann, 1990.

[31] C L Giles et al. Learning and extracting finite state automata with second order recurrent neural networks. *Neural Computation*, 1992.

[32] S Das, C L Giles, and G Z Sun. Learning context free grammars. In *PROCEEDINGS of the Cognitive Science Society*, 1992.

[33] T M Cover and J A Thomas. *Elements of Information Theory.* John Wiley and Sons Inc., 1991.

[34] R Garside. The CLAWS word-tagging system. In R Garside, G Leech, and G Sampson, editors, *The Computational Analysis of English: a corpus based approach.* Longman, 1987.

[35] A Voutilainen, J Heikkila, and A Antilla. *Constraint Grammar of English.* University of Helsinki, 1992.

[36] P. Pym. Perkins engines and publications. In *PROCEEDINGS of Technology and Language in Europe 2000.* DGXIII-E of the European Commission, 1993.

[37] S. Johansson and K. Hofland. *Frequency analysis of English vocabulary and grammar.* Clarendon, 1989.

[38] Yoh-Han Pao. *Adaptive pattern recognition and neural networks.* Addison Wesley, 1989.

[39] B Widrow and M Lehr. 30 years of adaptive neural networks. In C Lau, editor, *Neural Networks: theoretical foundations and analysis.* IEEE press, 1992.

[40] M W Goudreau, C L Giles, S T Chakradhar, and D Chen. First-order vs second-order single layer recurrent neural networks. Technical report, Princeton University and NEC Research institute, Inc, 1992.

[41] A K Chhabra, S Chandran, and R Kasturi. Table structure interpretation and neural network based text recognition for conversion of telephone company tabular drawings. In *International Workshop on Applications of Neural Networks to Telecommunications*, Princeton, 1993.

[42] S Roberts and L Tarassenko. A new method of automated sleep quantification. *Medical and Biological Engineering and Computing*, September 1992.

[43] J Shavlik, R Mooney, and G Towell. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 1992.

[44] G T Herman and K T D Yeung. On piecewise linear classification. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 1992.

[45] J Sklansky and G Wassel. *Pattern Classifiers and Trainable Machines.* Springer-Verlag, 1981.

[46] J-H Lin and J Vitter. Complexity results on learning by neural nets. *Machine Learning*, 1991.

[47] T M Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computing*, 14, 1965.

[48] J Hertz, A Krogh, and R Palmer. *Introduction to the Theory of Neural Computation.* Addison Wesley, 1991.

[49] D Rumelhart and J McClelland. *Parallel Distributed Processing.* MIT, 1986.

[50] L C W Dixon. *Nonlinear Optimisation.* English Universities Press Ltd., 1972.

[51] M L Brady, R Raghavan, and J Slawny. Back propagation fails to separate where perceptrons succeed. *IEEE Trans. on Circuits and Systems*, 1989.

[52] E D Sontag and H J Sussmann. Back propagation separates where perceptrons do. *Neural Networks*, 4, 1991.