Neural Nets for a Language Processing Task: Tag Disambiguation

Technical Report No.123

Caroline Lyon

January 1992

Neural Nets for a Language Processing Task : Tag Disambiguation

Caroline Lyon
School of Information Sciences

January 1992

Abstract

This paper first shows how part-of-speech tags can be ambiguous and why it is necessary to disambiguate them. Prototypes which can do this are developed in a limited natural language domain. The representation of syntactic data is discussed. An algorithm to disambiguate tags, using supervised training with a neural net, is presented. The single layer HODYNE net, which takes higher order input, is described, and its performance on this processing task examined. Using the simplest text up to 95% of tags can be successfully disambiguated, up to 88% in slightly more complex text. It is shown how altering the language representation and training parameters can affect performance. The results from Hodyne are compared to those obtained from a back propagation net with one hidden layer and found to be comparable, demonstrating that the higher order input data is linearly separable.

The work described here shows that there are syntactic patterns in natural language that neural nets can detect, and use for a language processing task.

1 The need to disambiguate part-of-speech tags

In spoken and written language the strings of words that occur are not random: there are likely sequences, recurring patterns and unlikely combinations. This implicit information can be extracted and used in language processing tasks in a number of ways.

First, probable sequences of words can be analysed. Secondly, individual words can be grouped into classes and an analysis made of sequences of class members. A third source of contextual information is derived from the structure of the language, which can provide information on likely constituents in a given context. Fourthly, semantic knowledge can be used to modify the likelihood of a word or phrase being used.

The analysis of sequences of individual words has worked well in restricted applications, such as the retrieval of information from a structured data base, with limited forms of enquiry. But in wider domains vast corpora of training data are required, and a significant proportion of word sequences are typically not covered. Examples that give an idea of the scale of the problem can be found in reports from IBM TJ Watson Research Centre [1]. For instance, 1.5 million words from a corpus of patent descriptions was analysed to find trigrams (sequences of three words). 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. It is suggested that a training set of 1 billion words is needed to get acceptable results from this method.

Another approach is to analyse words in groups, and use this information to predict whether a member of one class is likely to follow a member of another. The work described here adopts this method, and takes a system of classification based on (more or less) standard part-of-speech categories. The 12 to 18 categories used are described below in Section 3.

However, there is a problem with this approach, a problem particularly acute in English. This is that up to 1/3 of words typically have more than one part of speech tag, and simply looking up a word in a lexicon will give a choice of classifications. For example:

Look at a sentence like this.

The words "look" and "sentence" can be either verbs or nouns, "like" can be a preposition, verb or adverb, "this" can be a determiner or a pronoun.

An example of how this can undermine a processing task is seen where a language processor is used to support a speech recognition system [2]. Suppose an acoustic processor proposes several candidate words as possible matches for each discrete word of spoken input. Contextual information can be used to modify the possibilities of these candidate words being correct. The context analyser may take the candidates and see which belong to part-of-speech categories that are likely to provide successors to words already spoken. However, if words belong to multiple categories it is first necessary to find which is the right category, or tag, in the context.

One way to do this would be to apply rules that define what tag sequences are permissible. This approach is used to parse programming languages [3, 4]. Rule based systems have also been used in the natural language domain, described fully by Winograd in his much cited work (1983) [5] and also by Allen [6]. An early example is Woods' augmented transition network [7], equivalent to a context sensitive grammar, which is used in his LUNAR automated enquiry system [8]. Typically a rule based system implements a grammar which can produce all permissible sequences. However, the multiplicity of forms in natural language has proved a serious obstacle, unless the domain is carefully constrained. A second approach is to apply statistical probability criteria to disambiguate tags [9, 10, 11]. The third method, which is employed here, is to use a neural net to address the problem. These last two methods can both be considered "descriptive" as opposed to the "symbolic" rule based approach. Descriptive methods extract implicit information from corpora or texts of given language. This information is made explicit and used in language processing tasks.

The pattern matching capabilities of a neural net enable it to extract information on which sequences of tags are likely to occur. In order to derive this syntactic information a lexicon is first drawn up, giving possible part-of-speech tags to each word in the vocabulary. Some will have more than one. The text is decomposed into sentences, and each sentence generates a one to many mapping onto possible strings of tags. One of these has the correct tag allocation and is valid. These tag sequences are the raw input for a neural net. The neural net is trained, as described below, in supervised mode on sets of tag sequences so that it can classify new tag sequences as either valid or invalid.

2 Language Domain

These programs have been run on text from the Ladybird Reading Scheme books, levels 2 to 4. Vocabulary size ranges from 50 to 150 words. Samples of text are in the appendix. The domain of children's primers is appropriate for the tag disambiguation task since they include a high proportion of the most commonly used words. These are the sort of words that have been used for centuries and are likely to have multiple meanings and functions. Though the language is simple it is "real" in the sense that it was written for a purpose other than providing input for a language processor. Some other work in this field includes investigations of small samples of language, thought up for processing purposes [12], which it may not be possible to extend to an empirically significant scale.

The size of vocabulary is not necessarily an indication of complexity: this also depends on the structure of the language. The generally simple syntax of these texts make them a suitable medium for development. Text is decomposed into sentences and the work described in this paper

is limited to an investigation of patterns in linear sequences of words. More complex aspects of language structure, such as the grouping of words into clauses and phrases, are currently ignored. Using language from elementary reading primers minimizes this distortion. However, it does not eliminate the problem entirely. For instance from "Jack and the Beanstalk" comes the complex sentence

All we have is one cow.

The issue of representing language structure is discussed further below.

Much work in the language processing field has been done using texts from the LOB Corpus [9, 10, 13]. This is a 1 million word collection of texts from a number of different sources to give representative samples of different types of English. However, it is not an easy medium in which to begin developing prototypes. There is no restriction on vocabulary or linguistic structures, so difficult issues come in an overwhelming flood. And much larger amounts of text would need to be processed in order to extract patterns.

One drawback to using the Ladybird books is that the texts combine narrative and speech, which have different language patterns. For instance a narrative sentence would always have a verb, while a spoken sentence could have the form

"A fish" said Jane

In future work spoken and narrative language will be processed separately. Since speech in text is marked by quotation marks it is not difficult to separate the two types.

3 Forms of Representation

3.1 Input to the neural net

A neural net can be trained to recognize valid and invalid strings of part-of-speech tags in natural language. Now, the input to the neural net is a one dimensional vector. If the tags in a string are mapped directly onto the neurons of the input layer there is no preservation of word order. Therefore to capture the sequential nature of language data is entered into the input layer as tuples of adjacent tags - pairs, triples and pairs of pairs, as described below. This partially captures the sequential form of input. The ordering of tags within a tuple is preserved, though the the order of tuples within a sentence is not. The net takes a set of tuples and determines whether or not it is a valid set. If it is valid, the tag allocation is taken as correct.

There are a number of possible ways of partitioning a vocabulary into part-of-speech tag sets. It can be useful just to look at the two classes of "content" or "function" words [14, 15]. Content words are nouns, adjectives, verbs, adverbs that have semantic content, while function words are prepositions, conjunctions, auxiliary verbs etc. that link content words together. Content words are also known as "open class items", while function words are known as "closed class items" in traditional grammars. Whereas new content words can be added to the language the sets of function words are closed in the sense that they cannot normally be extended by the creation of additional members [16] There are many different tagging systems, ranging from those with 2 syntactic categories to others with more than 200 [9, 17].

It is also possible to partition a vocabulary in ways differing from the traditional part-of speech groupings. One approach is to classify words into categories determined by statistical analyses of their context. Words that appear in the same context as exemplar class models are put in the same category, and are known as "statistical synonyms"[18]. There has also been a recent revival of interest in much earlier work on radical reappraisals of the types of grammatical categories that can illuminate language [19].

However for this work more or less traditional part-of-speech categories have been used. The way in which text is classified into tag sets will have an important effect, and there are a number of relevant factors in making a choice. This research originated in a system to support a speech recognition device. The aim was to use contextual information to find a plausible part of speech to continue an incomplete sentence. For this reason a fine resolution was desirable to give as much syntactic information as possible.

On the other hand there were reasons to limit the number of tag classes. Prototypes were being developed and it was desirable to investigate the ability of nets to detect syntactic patterns with manageable amounts of data. Since the net is taking in tuples as input, the input space will be up to n^2 where n is the number of tag classes and the tag tuples are taken as pairs only. If they are taken as triples too the input space will increase to n^3.

Another reason to limit the number of tags was to enable patterns to be extracted more easily. Patterns imply repetition, and if there is a larger number of tag classes there will have to be a correspondingly larger quantity of training data to show up possible patterns.

Another factor was related to the use of the Hodyne net. This had previously been very successful in processing tag strings generated by a context free grammar with 9 tag classes. In applying it to a new natural language domain it seemed desirable to start on a similar scale.

Taking these factors into account the first investigations used a set of 12 tags. These were selected after inspecting the domain from which the text is taken. Later it was found that syntactic patterns could be distorted by compressing input into 12 categories for the more complex text of level 4 books, so the set was extended to 18. The differences in performance attributable to varying the tag set is discussed in Section 5.2 below.

3.2 Tag sets

The first set of 12 tags were

- Start symbol
- Determiner
- Noun
- Pronoun
- Verb - main
- Verb - auxiliary
- Infinitive "to"
- Preposition
- Conjunction
- Adjective
- Adverb
- Punctuation mark

In a set of tag classes of this size the infinitive "to" might not usually warrant separate treatment. But in these texts it occurred frequently and needed a class of its own. When level 4 text was inspected it was decided to extend these categories by adding the 6 categories, shown below with

examples. The category names are taken from Quirk and Greenbaum's "Grammar of English" [16].

The supplemental set of 6 tags were:

- Existential - *Here* they are.
- Predeterminer - *All* the children like the farm.
- Universal pronoun - We have *some* but they have not got *any*.
- Relative pronoun - They like the children *who* live on the farm.
- Possessive noun or pronoun - She takes *her* ball and *Peter's* boat.
- Speech marks - treated separately from other punctuation because they occur frequently.

It can be seen that many significant linguistic factors are ignored, such as singular and plural nouns, pronouns and verbs. Obviously syntactic information could be extracted from this data if it were used. However, though the omission reduces the amount of useful information that can be extracted, it does not produce incorrect information. The "noun - verb" sequence will still come up as a common pattern though it is not resolved as finely as possible. On the other hand, less frequent syntactic sequences can result in misleading information if the tagging scheme is too crude. For example if the word "all" is classified using the first 12 classes only some syntactic patterns will be disrupted. It would be designated a determiner, among other things, and then would upset the hitherto distintively false sequence "determiner - determiner" in sentences such as that starting "All the children ....".

4 The Processing Task

4.1 Mode of operation

In developing prototypes the programs are run in two modes:

1. The word tags are disambiguated for complete sentences at a time - "full sentence" mode. This is the initial development mode, and is used unless the second is specified.
2. The word tags are disambiguated for incrementing parts of a sentence, called "step" mode. Initial parts of the sentence are accepted as correct while they are an incomplete part of a full grammatical unit. This is the mode that models speech input, as for example in a speech activated word processor.

Nets are trained in supervised mode on a set of strings derived from sentences in one of the books. They are tested on sentences either from other text in the same book, or from another book, written with the same vocabulary.

4.2 Algorithm to disambiguate tags

An outline of the process by which tags are disambiguated is as follows:

1. The words in the vocabulary are assigned tags, possibly multiple. A maximum of two tags are used for these prototypes.
2. A piece of text is taken as training material. This text is decomposed into sentences.

3. Each sentence is represented by one or more strings of part-of-speech tags. When a word has more than one tag this gives rise to alternative strings. A sentence like

   The fish is in the water.

   has 2 words "fish" and "water" that could be verbs or nouns, so this will give rise to different strings of tags:

determiner	noun	verb	preposition	determiner	verb	punctuation
"	verb	"	"	"	"	"
"	noun	"	"	"	noun	"
"	verb	"	"	"	"	"

4. One of the strings corresponds to the textual sentence and is manually marked correct. The others are marked incorrect.

5. This set of correct and incorrect strings is used to train the net. The trained net is saved.

6. Now the trained net can be used. Other pieces of text from the same domain are input. The net will determine which combination of tag tuples in a sentence gives a correct string. The tags in these correct strings are taken as the right ones.

The training data should include a high proportion of near misses, rather than wildly ungrammatical sentences that are easier to detect. Using the process described above a sentence that has k words with multiple tags will generate k instances of strings that are near misses, i.e. with a single incorrect tag. The proportion of near misses to all the strings in the sentence is

$$k : 2^k$$

As sentences become more complex the number of alternative strings increases geometrically. Since the number of correct strings stays equal to the number of sentences, the ratio of incorrect to correct training strings shifts in favour of the former.

Sentences vary in length from 3 to 24 words, the median lengths ranging from 8 to 11 for different texts. The number of strings generated by a sentence will depend on the number of words with more than one tag. This will vary with the tag classification system used.

5 The Hodyne Net

5.1 Description of Hodyne

This research has been made possible through the use of the Hodyne net, developed by Wyard and Nightingale [20]. It was designed to distinguish between grammatical and ungrammatical sentences generated by context free grammars, and it accomplishes this task very successfully. It can also be effectively used in a natural language domain, as described here.

Hodyne is a single layer net, which takes as input contiguous pairs or longer tuples of data items. This gives the "higher order" part of its name, and partially captures the order of sequential speech or language input. A sentence is represented as a set of tuples of neighbouring parts-of-speech, and the net determines whether this is acceptable in the grammar. The architecture is simple: it has an input layer with weighted connections to the output layer. The "dynamic topology" is the optional ability of the net to create new nodes while training, so the size of the net can be minimised without knowing the extent of the input data in advance. This original facility was in fact subsequently dropped, in order to speed up processing time. It is now assumed

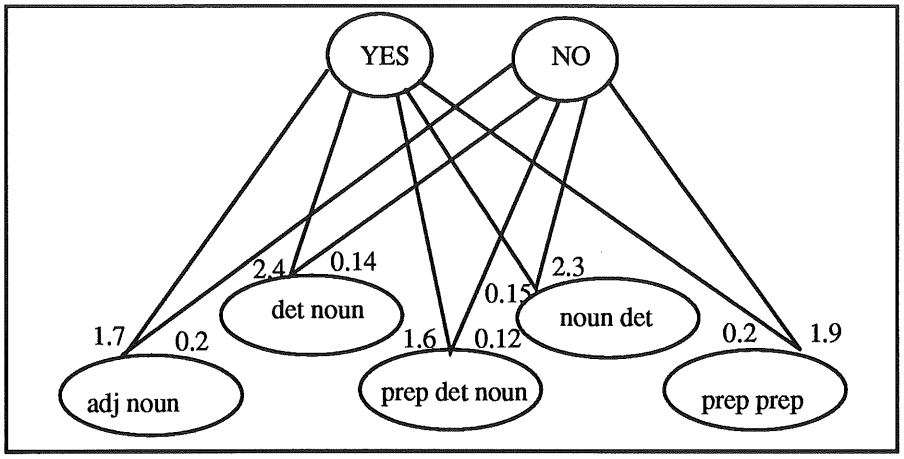

Figure 1: The architecture of HODYNE, showing a portion of the net trained on the grammaticality problem, with typical values for weights. Reproduced by permission.

that all possible input nodes are known in advance. There are two output nodes signifying "yes" and "no" for valid and invalid strings. The node with the greatest activation fires.

While this work was in progress Hodyne was modified so that it would run more quickly. Training time was improved by more than a degree of magnitude. This was achieved in two ways : first, the process was streamlined and a number of experimental features were dropped. Secondly, the representation of the input layer was changed from a linked list to a hashed array. In the earlier version of the program processing was delayed by the time taken to scan across linked lists of nodes to find matches.

As an example of the operation of Hodyne consider part of a net trained, in supervised mode, on sentences generated by the context free grammar and on sentences with deliberate faults. The sentences translate into part-of-speech strings as follows:

the black cat sleeps in the sun.....det adj noun verb prep det noun
black cat the sleeps in the sun.....adj noun det verb prep det noun
the black cat sleeps by in the sun.....det adj noun verb prep prep det noun

This could produce a net part of which is shown in figure 2. Weights are initially 1.0, and are adjusted, as described later, as the net is trained.

The net is not necessarily fully connected between input and output layers. When an input node fires, representing a tuple present in a given sentence, a connection is made, if it is not there already, only to the desired response node. Thus, a node representing a tuple which is always illegal will not be connected to the output node named "valid". However, a node representing a tuple that can be legal can be connected to both the "valid" and the "invalid" output node since it can appear in both a legal and an illegal sentence.

The learning strategy can be described as "connect", if not already connected, and "punish to teach". There is no reward for a correct response but weights are adjusted when an incorrect answer is produced. In this case the active input nodes are selected, and the links to the wrong answer node, which has fired, are decremented. Thus input nodes that contributed to the wrong answer have their weight on that link reduced. The links to the desired response node are correspondingly incremented. The update function used to alter the weights is the following:

$$w_{new} = \left[1 + \frac{\delta w_{old}}{1 + (\delta w_{old})^4}\right] w_{old}$$

where δ= +1 for strengthening weights and δ = -1 for weakening them.

This function satisfies two constraint requirements. Firstly, weight changes are bounded, and the factor by which an old weight is multiplied to give the new weight tends towards 1 as the weights themselves move away from a central value. The update function is asymptotic to 1, eventually becoming saturated. Secondly, any change is attenuated, so that a single counter example does not have undue influence on the weights.

The most effective method of training was found to be incremental. Using this approach training commences on a small group of training strings, and when performance is satisfactory the next group is added until the complete training set has been covered.

Once the net is trained it is tested on unseen data. The weights are of course fixed and the net acts as a function evaluator, returning " yes" or "no" for a given input.

The classical problems with single layer nets were concerned with first order input. The limitations for first order nets that Minsky and Papert exposed do not apply in the same way to higher order input. A simple illustration can be found in [21, p 23]. Higher order input to neural nets is an effective form of representation whenever temporal sequences or invariant relationships between features need to be captured [22]. Further investigation into linear separability is possible using linear programming techniques described by Mangasarian[23, 24].

5.2 Performance of Hodyne

5.2.1 Overview

Using the simplest text from level 2 books, nets trained in full sentence mode can determine correct tag allocation in up to 95% of cases. However, as slightly more complex text at level 4 was used the best results were 88%. The texts from level 2 books have minimal syntactic complexity, and are closer to texts generated by context free grammars for which Hodyne was originally designed than less restricted language.

The programs were run altering the following variable factors as well as the input source:

- type of tuples input - pairs only, pairs and triples, or pairs of pairs.
- number of tags - 12 or 18
- mode of processing - "full sentence" or "step"

The following training parameters were also varied:

- the size of the training set
- the threshold for acceptance of training data
- the size of the subsets for incremental training

5.2.2 Effect of using different tuple types

Using pairs as input gives as good, or better, results than higher order tuples. Performance did not improve when the programs were run using triples. Pairs of pairs were also input, for the

smaller tagset of 12 only, which again did not improve performance. Details are shown in Table I. It seems that the larger input space produces too sparse data, and that there is a much lower probability of triples or quadruples recurring. It would be necessary to investigate much larger training sets to elicit patterns of triples. In section 7.1 another method of data representation is described that might get round this problem.

Table I: To illustrate the effect of using different tuple types

Text	tuple type	training set size	test set size	nodes generated	% correct training	% correct test
level 2	pairs	326	89	70	96	95
	pairs and triples			285	98	93
	pairs of pairs overlapping*			228	98	91
	pairs of pairs adjacent*			269	97	83
level 2	pairs	260	89	64	97	94
	pairs and triples			257	98	85
level 4	pairs	347	120	123	96	88
	pairs and triples			518	98	81

* Pairs of pairs overlapping group input elements a b c d as p[(a,b),(b,c)] [(b,c),(c,d)] etc. Pairs of pairs adjacent group these elements as [(a,b),(c,d)][(c,d),(e,f)] etc.

5.2.3 Effect of using different tagging schemes

Intuitively, it seems probable that there is a relationship between the complexity of the text and the tag set that gives optimal results. These programs indicated that this could possibly be the case. For level 2 books 12 tags gave as good results as 18. But for level 4 books 18 tags were typically best, though occasionally there was no improvement. The extra tags were mainly used to handle function words, so that distinct tag sequence patterns were separated. Table II illustrates the effects of altering the tag set.

Table II To illustrate the effect of using different tag sets. (The same text produces different size input sets for different tag sets.)

Text	tag set size	tuple type	training set size	test set size	% correct training	% correct test
level 2	12	pairs	326	89	96	95
	18		340	94	95	90
level 2	12	pairs and triples	326	89	98	93
	18		340	94	98	93
level 4	12	pairs	315	122	95	82
	18		347	120	96	88
level 4	12	pairs and triples	315	122	99	80
	18		347	120	99	81
level 4	12	pairs	421	194	91	85
	18		441	212	90	85
level 4	12	pairs and triples	421	194	96	78
	18		441	212	97	84

5.2.4 Processing in "step" mode

Most of the work described in this paper was conducted in whole sentence mode. In step mode there is less context to provide pattern information, particularly at the start of a sentence when a single word has been input. However, results up to 85% correct are still obtained at level 2, and up to 82% correct at level 4.

The preprocessing of data to present to the net is a much longer task in step mode than in the whole sentence mode. A given piece of text will generate many more incrementing parts of a sentence to be used as input data. A sentence with n words without any ambiguous tags will produce n incrementing parts. With ambiguous tags this figure will of course be multiplied, its size depending on the position in the sentence of the words with more than one tag.

Some experiments were made testing text in step mode on nets trained both in step and in whole sentence mode, and the latter performed as well as the former.

Table III To show the results of testing text in "step" mode.

Text	training set type	training set size	test set size	number of presentations	% correct training	% correct test
level 2	step	440	1020	20,000	95	85
		1020	440	106,000	97	81
level 4	full	347	654	22,000	96	81
	step	966	654	47,000	93	81
	full	416	654	420,000	95	82 *

* See comment below on this result.

5.2.5 Size of training sets

The size of training sets was initially taken rather arbitrarily between 250 and 500 for whole sentence mode. In fact these were coherent sections of text, and this range turned out to be appropriate. Below this size performance degrades. Above this size there was a problem training the net to a threshold of 90% or more. This arose because the function for updating weights starts to approach 1 as weights increase or decrease outside a certain range. When the size of the training set increases this saturation mechanism comes into play as some of the weights will tend to move into this range. The weight adjustment routine is repeated as the update function cannot alter the weights sufficiently, and the process gets stuck.

However,the net may still be trained on the first part of the training set, and give good results on test data. An example is the last entry in Table III, marked " * ". In this case the training process stuck when 416 out of 666 strings had been trained. At this point training was aborted. However, when this net was tested the results were marginally better than for the other nets. When this net was used on level 4 text in full sentence rather than step mode 90% correct results were obtained. This performance was slightly better than that cited elsewhere in this paper, but it was felt that more investigation of this result was necessary.

In step mode the same number of sentences generated far more strings, since incrementing parts were used as input data. The average string length is of course much shorter, and the problem of reaching saturation did not arise with a training set size over 1000.

5.2.6 Threshold of acceptance for training

The threshold for acceptance was varied between 90% and 97%. The training set typically trained to several percentage points above the chosen threshold. When the threshold was set high the training set would reach a higher level of correct results (occasionally it would fail to train above

95%). However, the results for the test set tended to deteriorate. This is an example of a net "overtraining". The net is too well fitted to the training data and is less well able to generalize correctly on unseen data, unless the training data represents the whole population very well. Though we did not operate a threshold selection procedure, this can be done using a validation set [25]. This is a set of input strings separate from both the training and the test set. It is run as a "trial test" on the net trained to various thresholds to determine the optimal level.

5.2.7 Training time and size of incremental training step

The time taken for a net to train was not a restricting factor with the latest version of Hodyne. Running on an Apollo workstation the speed varied from about 3000 to 10,000 presentations a minute of CPU user time. The time taken for each presentation depends on the length of the string and the number of input nodes which have to be searched. The number of presentations needed to train a net was typically in the range 10,000 to 25,000, though of course it could extend indefinitely for a net that would not train.

The incremental training step size was varied from 1 to 10, but it was found that with an incremental step of 5 strings the nets trained effectively. This was appreciably faster than training with single increments, as far fewer presentations were necessary. For instance, using a step size of 1, 53,000 presentations were necessary to train a net that returned 93% correct on test data. Using the same training set, threshold and achieving the same result on test data, only 11,320 presentations were needed with a step size of 5. The optimal training step size has not yet been investigated further.

6 The Back Propagation Net

6.1 Description of back propagation model

Similar programs were run on back propagation networks with one hidden layer, based on the model described by Rumelhart et al. [26]. The algorithm used, described in section 4.2 above is the same. The input into the net is a vector whose elements represent all possible tag pairs. For a 12 tag classification the input vector will have 144 elements, initialised to 0. When a string of tags is generated this will be mapped onto the input vector by flagging to 1 each tagpair that occurs. Note that this representation is the same as that used by Hodyne, but tuples of order higher than 2 were not investigated. Level 2 texts with 12 tags and level 4 texts with 18 tags were used.

The net has three layers: an input layer with 144 (12^2) or 324 (18^2) nodes, an output layer with 2 nodes and a hidden layer in which the number of nodes was varied in different trials. Each layer is fully connected to the next. In order to train the net weights on connections are first initialised randomly between -1 and +1. The training input is fed through to the output, using a sigmoid transfer function. At the output nodes the error is found, scaled by the derivative of the transfer function to give the "delta weight" and propagated back. The weight on a connection is then modified in the usual way, by an amount proportional to the current delta weight plus a momentum term proportional to the previous delta weight.

Learning coefficients used are 0.3 for the input to hidden layer, 0.15 for the hidden to output layer. A momentum coefficient of 0.4 is used. It typically took 10,000 to 20,000 presentations to train a net. With 72 hidden nodes 1000 presentations took about 2 minutes, with 2 hidden nodes 1000 presentations took about 15 seconds.The programs are run on a Sun Sparc 1+ workstation using Neuralworks II.

6.2 Performance of back propagation nets

As Table IV shows, results are comparable to those for Hodyne. At level 2 the net was trained on one book and tested on text from a parallel book. The training and test sets were then reversed. The results are significantly better when the larger set is used for training.

Table IV To show the results of using back propagation nets (BP), and compare to Hodyne. All tuple types are pairs.

net	text	tag set size	training set size	test set size	hidden nodes	% correct training	%correct test
BP	level 2	12	326	264	72	96	91
					9	97	93
					6	97	92
					1	95	92
Hodyne						95	90
BP	level 2	12	264	326	72	99	81
					36	99	81
					18	99	81
					9	99	82
					6	99	80
					1	99	84
Hodyne						96	84
BP	level 4	18	347	120	9	99	82
					6	99	80
					2	99	83
					1	95	89
Hodyne						96	88

The architecture of the net was varied by altering the number of hidden nodes. This made very little difference to the results even when the number of hidden nodes was reduced to one, as is expected with linearly separable input. The activation of the two output nodes complement each other, and the same results can be obtained with a single output node.

Examining individual cases where there is a wrong result it is found that up to 50% of the errors on Hodyne match errors on the back propagation net. This suggests that the training sets generalise well, but as they are taken from samples of natural language they do not entirely cover the test set.

7 Discussion

7.1 Data representation

It has not been possible so far to utilize the information in triples and higher order tuples. The combinatorial explosion of the size of the input space means that the input data is too sparse. However, another method of data representation could possibly circumvent this problem. What is needed is a method of compressing information from higher order tuples into a more compact input space.

One way to do this is to represent data in a form such that elements can be combined by an operator that is associative but not commutative. Thus

$a * b \neq b * a$ but $a * (b * c) = (a * b) * c$

The sequential nature of input is captured, but tuples of higher orders can be combined so

that they map onto a data item that is the same size as a single element. One interpretation is to represent input elements as matrices and use the matrix multiplication operator. This method has been successfully applied to phonemes by Lucke and Fallside [27].

7.2 Language structure

So far many aspects of the structure of language are ignored. In particular words could be grouped into syntactic elements like clauses and phrases. When this is ignored the relationship of adjacent words at clause and phrase boundaries can upset the general patterns. Consider a sentence like

> The boys who were laughing walked past.

At the end of the clause "who were laughing" the words "laughing walked" are adjacent, but "walked" is the verb whose subject is "the children". In these cases distant constraints are more important than neighbouring ones. We would like to be able to group words together in blocks so that elements like clauses and phrases could be used as data items. We would also like to implement a stack mechanism, so that phrases and clauses of variable length could be pushed out, like subroutines in a computer program, and processed separately.

A variation on this problem is addressed in research into ways of modelling recursion with neural nets [28, 29]. However, most of the ongoing work is concerned with modelling embedded strings of set lengths. In natural language we want to be able to identify and extract strings of varying lengths. The issues on which this touches have been studied extensively by linguists for many years, sparked off by Chomsky's seminal work in 1957 [30].

8 Conclusion

This work shows that there are syntactic patterns in natural language that neural nets can detect. Higher order input of tag tuples seems to be linearly separable, since a single layer net gives as good results as a back propagation model. In the limited domain of childrens' books the Hodyne net has been successfully used to disambiguate tags. Future work will extend into larger vocabularies and more complex language. The issue of representing groups of words as phrases and clauses will be addressed.

Acknowledgements

Caroline Lyon is supported by a grant from the Science and Engineering Research Council.

Dr Charles Nightingale and Peter Wyard of BTL, who originally developed the Hodyne net, are thanked for their help and encouragement. Dr S L Stott, Professor L C W Dixon and J A Davis are also thanked for their support.

Appendix

Samples of text : Level 2

Here is a tree. Pat looks into the tree. He looks for Peter and Jane. They are in the tree. Look here, says Jane. Look here, Peter. Come and look. Peter comes and they look. Here are some sweets. Some are for Peter and some are for Jane. They have some sweets. They like sweets.

Peter wants to jump and Pat wants to jump. They jump for fun. Can you jump this? says Peter to Jane. Yes, says Jane. Yes, I can jump this. I want to jump. Look, Peter, look. I can jump this.

Peter and Jane go to the shop. They go into the shop for some fish. Peter has the dog and Jane has the fish. The fish go into the water. Into the water they go. Pat wants the fish. No, Pat, no, says Peter. No, no, no, says Jane to the dog.

Level 4

The two children are at the farm. They like to work there. Here they are with the horses. Jane likes her little horse. She gives it an apple. She wants to keep her little horse.

Peter has a big horse. "I want to get on my horse, Jane," he says."Help me up, please." Jane helps Peter to get on his big horse. "There you are," she says. "Away you go." Then Jane gets on her little horse. "Away I go, on my horse," she says.

The children go off to work on the farm. "Let us help with the cows," says Peter. "Yes," says Jane "We will help with the cows." "Let us help the man milk the cows,"says Jane. "Will he let us help him milk?" "Yes," says Peter. "He likes us to help him with his work." "Yes," says the man. "You two can help me with the milk. Put the horses away. Come and help me with the cows."

References

[1] F Jelinek. Self-organized language modeling for speech recognition. In *Readings in Speech Recognition.* Morgan Kaufmann, 1990. IBM T.J.Watson Research Centre.

[2] C Lyon and R Frank. Improving a speech interface with a neural net. In R Beale and J Finlay, editors, *Neural Networks and Pattern Recognition in Human Computer Interaction.* Ellis Horwood, 1992.

[3] A V Aho and J D Ullman. *The Theory of Parsing, Translating and Compiling.* Prentice Hall, 1972.

[4] D E Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 1968.

[5] T Winograd. *Language as a Cognitive Process.* Addison Wesley, 1983.

[6] J Allen. *Natural Language Understanding.* Benjamin Cummings, 1987.

[7] W A Woods. Transition network grammars for natural language analysis. In B L Webber B J Grosz, K Sparck-Jones, editor, *Readings in Natural Language Processing.* Morgan Kaufmann, 1970.

[8] W A Woods. Semantics and quantification in natural language question answering. In B L Webber B J Grosz, K Sparck-Jones, editor, *Readings in Natural Language Processing.* Morgan Kaufman, 1978.

[9] R Garside G Leech and G Sampson. *The Computational Analysis of English.* Longman, 1987.

[10] C G de Marcken. Parsing the lob corpus. In *Association of Computational Linguistics*, Pittsburgh, PA, 1990.

[11] J H Wright. L r parsing of probabilistic grammars with input uncertainty for speech recognition. *Computer Speech and Language*, 1990.

[12] G W Cottrell. *A Connectionist Approach to Word Sense Disambiguation.* Pitman, 1989.

[13] S Hanlon and R Boyle. Syntactic knowledge in word level text recognition. In R Beale and J Finlay, editors, *Neural Networks and Pattern Recognition in Human Computer Interaction.* Ellis Horwood, 1992.

[14] Kai-Fu Lee. Context dependent phonetic hidden markov models for speaker independent continuous speech recognition. *IEEE Trans. on Acoustics, Speech and Signal Processing*, April 1990.

[15] F Mosteller and D L Wallace. *Inference and Disputed Authorship: The Federalist.* Addison Wesley, 1964.

[16] R Quirk and S Greenbaum. *A University Grammar of English.* Longman, 1976.

[17] E Atwell. *LOB Corpus Tagging Project.* University of Lancaster, 1982.

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

[19] B L Whorf. *Language, Thought and Reality.* MIT, 1956, 1937.

[20] 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.

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

[22] M B Reid L Spirkovska and E Ochoa. Rapid training of higher-order neural networks for invariant pattern recognition. In *Proceedings of the International Joint Conference on Neural Nets*, Washington, DC, 1989.

[23] O L Mangasarian. Multisurface method of pattern recognition. In *IEEE Transactions on Information Theory*, 1968.

[24] K P Bennett and O L Mangasarian. Neural network training via linear programming. Technical report, Center for Parallel Optimization, University of Wisconsin, 1990.

[25] R F Harrison S J Marshall and R Kennedy. Neural classification of chest pain symptoms: a comparative study. In *IEE Conference on Artificial Neural Nets*, 1991.

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

[27] H. Lucke and F. Fallside. Application of the compositional representation to lexical access using neural networks. In *ICSLP , Kobe*, 1990.

[28] M Zeidenberg. *Neural Networks in Artificial Intelligence.* Ellis Horwood, 1990.

[29] J Pollack. Recursive auto-associative memory: Devising compositionaldistributed representations. *New Mexico State University*, 1988.

[30] N Chomsky. *Syntactic Structures.* Mouton, The Hague, 1957.