

# Automated Language Processing with Neural Networks: the ALPINE partial parser

Caroline Lyon

C.M.Lyon@herts.ac.uk

Department of Computer Science, University of Hertfordshire

## Abstract

This paper describes the Alpine parser, a hybrid, corpus based processor in which neural networks operate within a grammatical framework. The purpose of the parser is to take straightforward English text, and decompose declarative sentences into syntactic constituents. It first finds the position of the subject. This decomposes the sentence into three concatenated segments, *pre-subject - subject - predicate*, which can then be further parsed independently. This reduces the complexity of the parsing task.

The neural processing component belongs to the class of Generalized Single Layer Networks. Such networks offer advantages of functional transparency and operational speed. In this parser, the initial stage of processing maps linguistic data onto a higher order representation, which can then be analysed by a single layer network. This transformation is supported by information theoretic analysis.

The parser has been trained and tested on text from technical manuals. It also works quite well on straightforward English from other domains, and there is a prototype on which you can try your own text.

## 1 Introduction

The pattern matching powers of neural networks can be used to detect syntactic patterns in text. This paper describes the Alpine parser, a hybrid, corpus based processor in which neural networks operate within a grammatical framework. A prototype is available, on which you can try your own text. The core of this work has been published in the neural computing literature (Lyon and Frank, 1997). We want to present it for the scrutiny of the language processing community too.

The purpose of the parser is to take ordinary English text and decompose declarative sentences into syntactic constituents. It first finds the subject of the sentence and the pred-

icate. Declarative sentences can almost always be segmented into three concatenated sections

*pre-subject - subject - predicate*

as shown in Figure 1. Other constituents, such as clauses, phrases, noun groups are contained within these segments, but do not normally cross the boundaries between them. Here are three examples from a manual, with the subject delimited by square brackets:

[ There ] are two methods of accelerating searching.

[ Your ability to use either search method ] is determined by your familiarity with your system.

To use the first type of accelerated search, [ you ] must know the search menu line number for the type of search you want.

Though a constituent in one section may have dependent links to elements in other sections, such as agreement between the head of the subject and the main verb, once the three sections have been located they can be partially processed separately, in parallel. This hierarchical decomposition reduces the complexity of the parsing task. For instance, after the tripartite division of the sentence, the Alpine parser goes on to find the head of the subject, as shown in the examples in the Appendix.

The processor has been trained on text from technical manuals, but because syntactic patterns are so pervasive it can be used on a wider range of straightforward English text.

The paper is organised in the following way. First we look at some characteristics of English language. and its representation. Next we give a description of the Alpine parser and show how it works. Then we take an overview of neural network design, and explain how design decisions can be approached. Finally we conclude with some measures of Alpine's performance.

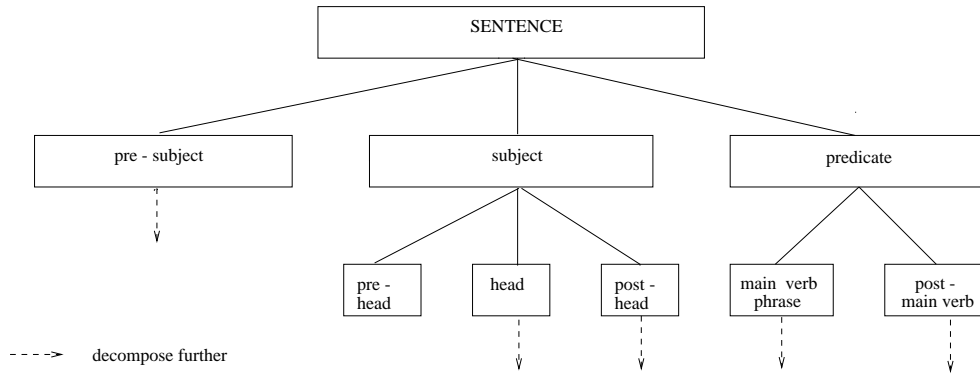


Figure 1: Decomposition of the sentence into syntactic constituents.

## 2 Characteristics of English Language Data

The significant characteristics 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

### 2.1 The distinctive distribution of linguistic data

The distribution of words in English and other European languages has distinctive characteristics, to which Shannon drew attention (Shannon, 1951). He indicated the extent to which a significant number of words occur infrequently. For example, words that have a frequency of less than 1 in 50,000 make up about 20-30% of typical English language news-wire reports (Dunning, 1993). This problem of sparse data means that automated speech recognition systems need very large corpora for training (Gibbon et al., 1997, page 404).

In order to address the problem of an indefinitely large vocabulary together with the sparse data phenomenon words are mapped onto part-of-speech tag classes. This also make syntactic patterns more pronounced. For the first stage in processing, finding the subject of a sentence, Alpine uses a tagset of 19 part of speech tags. It uses 26 for subsequent processing.

### 2.2 Grammatical structure

There is an underlying hierarchical structure to the English language, a phenomenon that has been extensively explored. Declarative sentences will usually conform to certain structural patterns, as is shown in a simplified form in Figure 1. Text and speech also, of course, contain non-declarative sentences: imperatives, questions, exclamations. They also contain non-sentential elements such as headings, captions, tables of contents. The work described in this paper is restricted to declarative sentences.

Within the grammatical structure there is an indefinite amount of variation in the way in which words can be assembled. 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)

### 2.3 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. After finding the position of the subject, the Alpine parser goes on to find the head of the subject, as in sentence (3) below.

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

The number of the subject is determined by its head “connections”, which is not adjacent to the verb “must be” : The subject of this sentence is plural. Note that in English modal verbs like “must” have the same singular and plural form, 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.

In our system, the neural classifier will pick up local dependencies. The rule based, grammatical framework will give us the ability to find some long distance dependencies within sentences.

### 3 Language Representation

By mapping an indefinite number of words onto a limited number of part-of-speech tags we retain the ability to find syntactic components, though of course we lose semantic information.

As well as mapping words onto tags, we need to capture a representation for the boundaries of the syntactic component we are looking for, the subject initially. We do this by introducing “virtual tags” or hypertags as boundary markers. They are represented by ‘ [ ’ to mark the start of the subject and ‘ ] ’ to mark its close. The task of the neural processor is to find the correct placement of these markers.

#### 3.1 Modelling discrete sequential data

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 (Sejnowski and Rosenberg, 1987).

Another method is the use of recurrent nets (Elman, 1991). In its basic form this type of network is equivalent to a finite state automaton that can model regular languages (Giles, 1992).

The third approach, used in this work, is the the n-gram method. We take sets of ordered, adjacent elements, which capture some of the sequential structure of language. This is related to the well known trigram approach used in automated speech recognition (Gibbon et al., 1997).

Part-of-speech tags have relationships with each other. For example (preposition, verb) is unlikely to occur, while (noun, verb) is a quite frequent combination. In the same way, tags have probabilistic relationships with hypertags: ( [ , verb) cannot occur, while ( [ , determiner) is common.

One of the advantages of using neural processors is that they can conveniently model “improbabilities” as well as probabilities.

That this method of representation captures some of the structure of natural language, is shown by analysis with information theoretic techniques. The entropy of a sequence of tags declines when pairs and triples are used. Entropy can be understood as a measure of uncertainty. The uncertainty about how a partial sequence will continue can be reduced when information about neighbouring elements is taken into account. Shannon introduced this approach by analysing sequences of letters (Shannon, 1951), where the elements of a sequence are either single letters, or pairs of adjacent letters, or triples. These are n-grams, with  $n$  equal to 1, 2 or 3. The entropy of a sequence represented by letter n-grams declines as  $n$  increases. When sequences of tags were analysed the same result was obtained: the entropy of part-of-speech n-grams declines as  $n$  increases from 1 to 3.

### 4 The Alpine parser

For this paper, we explain how the subject of a declarative sentence is detected. Similar methods are used for further processing. We take a declarative sentence, and the task of the parser is to correctly place the hypertags marking the subject. This is the process.

First, map each word onto one or more tags.

Then generate strings with hypertags placed in all possible positions, for all combinations of tags. This of course produces an unmanageable amount of data, so a pruning process is integrated into the string generation. Applying local and semi-local constraints, the generation of a string is zapped if a prohibited feature is produced. A prohibited feature is an element in the Prohibition Table, which is set up in advance. An example would be the pair (verb, verb). Of course, (auxiliary verb, verb) is all right, as is the triple (verb, ] , verb)<sup>1</sup>.

---

<sup>1</sup>For example: “ [ Those who ran ] arrived first.”

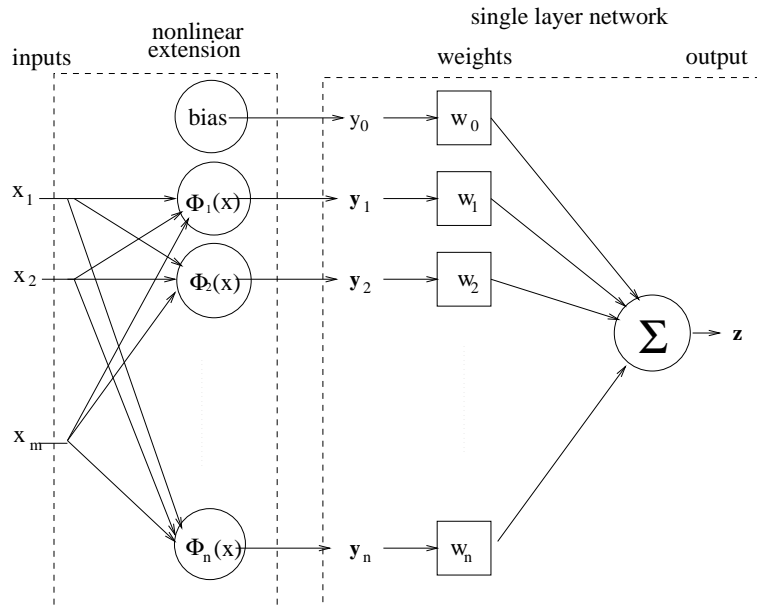


Figure 2: The Generalized Single Layer Network, GSLN, with 1 output

Then each string is taken in turn, and the tags are combined into pairs and triples (tuples). Thus each string is represented by a set of tag tuples.

The input to the neural network is a binary vector, whose elements represent these tuples. If a tuple is present in a string, the element that represents it in the input vector is changed from 0 to 1.

#### 4.1 Coding the input

As an example of input coding consider this short sentence:

Papers published in this journal are protected by copyright. ....(4)

(A) Map each word onto 1 or more tags

|           |                       |
|-----------|-----------------------|
| papers    | verb or noun          |
| published | past-part-verb        |
| in        | preposition or adverb |
| this      | pronomial determiner  |
| journal   | noun                  |
| are       | auxiliary-verb        |
| protected | past-part-verb        |
| by        | preposition           |
| copyright | noun                  |
| .         | endpoint              |

(B) Generate strings with possible placement of subject boundary markers, and possible tag allocations (pruned).

Exemplars:

```
strt [ verb ] pastp prep prod noun aux pastp
prep noun end
strt [ noun ] pastp adv prod noun aux pastp
prep noun end
strt [ noun pastp ] adv prod noun aux pastp
prep noun end
strt [ noun pastp prep prod noun ] aux pastp
prep noun end *** target ***
strt [ noun pastp adv prod noun ] aux pastp
prep noun end
```

Each string represents a possible placement of the subject boundary markers.

In training mode the correct string will be identified. In test mode, the processor will have to find the correct string.

(C) The next step is to take the tag string and map it onto a set of tag pairs and tag triples (tuples). These will be the input into the network.

Exemplars:

```
(strt, [ ] ( [ , verb ) ( verb, ] ) ..... (noun,
end)
(strt, [ , verb ) ([ , verb, ] ) (verb, ], past-part)
..... (prep, noun, end)
```

and similarly for other strings

The input for the neural network is a binary vector, with each element representing a tag pair or triple. A given sentence will activate some of these binary elements. In training mode the weights in the network will be adjusted until the desired output is obtained. In testing mode the output for each string is found, and the string with the highest “grammaticality score”, symbolised as  $\Gamma$ , is the “winner”. The placement of the hypertags in the winning string is accepted.

## 5 Neural network design

We have used supervised, feed forward networks in our work. A good clear introduction to these types of networks is given by Tarassenko (Tarassenko, 1998).

This prototype is based on a single layer network, which can only process data that is linearly separable. The more general model is a 2 layer network, such as the multi-layer perceptron (MLP) which can process data that is not linearly separable.

However, 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 (Pao, 1989, chapter 8). Whether this is desirable in any particular case must be investigated. The enhancement can map the input data onto a space, usually of higher dimensionality, where it will be linearly separable.

Figure 2 illustrates the form of the Generalized Single Layer Network (GSLN). This figure is derived from Holden and Rayner (Holden and Rayner, 1995). A non-linear transformation  $\Phi$  on inputs  $\{x\}$  converts them to elements  $\{y\}$ , which a single layer net will then process to produce an output  $z$  (temporarily assuming 1 output).

The  $\Phi$  functions, or basis functions, can take various forms. They can be applied to each input node separately, or, as indicated in the figure, they can model the higher order effects of correlation. In our processor  $\Phi$  is an ordered ‘AND’. A similar function is used in DNA sequence analysis (Lapedes et al., 1992). The  $\Phi$  function can be arithmetic: for instance, for polynomial discriminant functions the elements of the input vectors are combined as products

(Duda and Hart, 1973, page 135).

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 a single layer net is used for the second stage of processing.

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.

### The conversion function

We use this approach in the parser by converting a sequence of tags to a higher order set of adjacent pairs and triples. The example in section 4.1, stage C shows how the input elements are constructed.

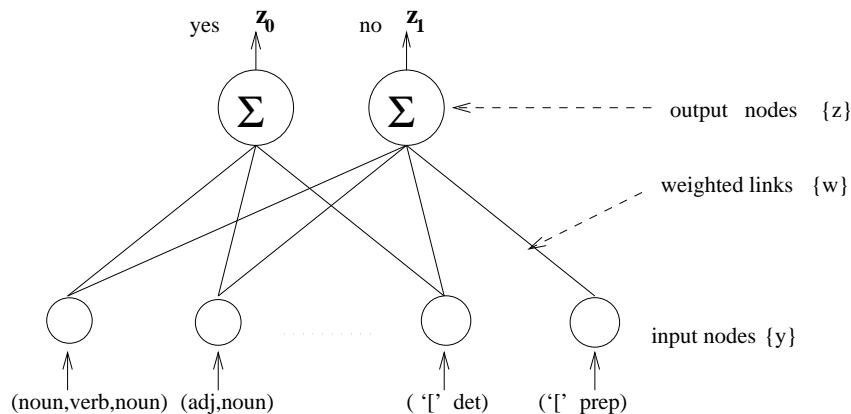
This can be related to Figure 2. Let each  $x_i$  for  $i = 1$  to  $m$  represent a tag. The  $\Phi$  functions map these onto  $m^2$  pairs and  $m^3$  triples, so  $n = m^2 + m^3$ . Only a subset of  $n$  will be activated.

## 6 Operation of the neural network

Experiments were conducted on a number of different architectures, but best results were obtained using the Hodyne network, first introduced by Wyard and Nightingale (Wyard and Nightingale, 1992). This network is shown in Figure 3. The 2 outputs  $z_0$  and  $z_1$  represent grammatical and ungrammatical, “yes” and “no”, results. In training, a grammatical string must produce  $z_0 > z_1$ , and vice-versa, else a weight adjustment is invoked.

The network is not fully connected. A link between an input element and an output node is only activated when that element appears in an input string. The link to the desired output will then be activated. Of course, many tag tuples occur in both grammatical and ungrammatical sentences, and are connected to both output nodes. More implementation details can be found in (Lyon and Frank, 1997).

With single layer networks we can use a direct update method: if a tagstring produces the wrong output, then the weights connecting it to the output nodes are adjusted. If the classification is correct, then the weights are left alone. The weight adjustment can be a function of the



'[' represents the start of the subject. The node ('[' determiner) would occur often in both correct and incorrect strings. The node ('[' preposition) would not occur in a correct string, so it is not connected to the "yes" output node.  $\Sigma$  represents summing function.

Figure 3: The Hodyne network.

input vector, as in the Perceptron, or of the existing weights, as in the network used here. It is also possible to use an error minimization method, based on the difference between target and actual output, using differentiable activation functions to determine the weight adjustment. This is the basis of the standard multi-layer back propagation network, and can also be used in a simplified form for single layer networks. Experiments using this method did not give such good results as the direct update approaches.

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. For this network the grammaticality measure  $\Gamma$  is  $z_0 - z_1$ .

## 7 Performance

This work has principally been developed on text of technical manuals from Perkins Engines Ltd. (Pym, 1993), and the ALPINE prototype was trained on 351 sentences from these manuals, as shown in Table 1. The prototype has then been run on sentences from two other corpora from technical manuals, described in (Sutcliffe et al., 1996). They are taken from the Dynix Automated Library Systems Searching Manual, and the Trados Translators Workbench for Windows User's Guide.

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

Table 1: Corpus statistics

In analysing the performance we will consider whether the subject boundary markers are correctly placed. Other metrics include an analysis of tag disambiguation, and measures of whether other constituents are correctly found.

All declarative sentences were extracted from some of the Perkins manuals. 2% were edited so that they fell within the restrictions on subject and pre-subject length. The Alpine processor cannot handle sentences with pre-subject 15 words or more, subject 12 words or more. Punctuation marks count as words.

The 351 sentences that made up the Perkins data was divided into four parts. The neural networks were trained on three of the parts and tested on the fourth. The performance of the neural networks are given below in Table 2.

After these tests the Hodyne network was then trained on all the Perkins data, augmented with a small amount of other straightforward

| Ratio test set / training set | % hypertags correctly placed |
|-------------------------------|------------------------------|
| 0.10                          | 100                          |
| 0.20                          | 100                          |
| 0.23                          | 100                          |
| 0.26                          | 95.5                         |

Table 2: Results on Perkins data, after 2% sentences had been edited out, leaving 351 sentences.

| Text   | Number of sentences | % hypertags correctly placed |
|--------|---------------------|------------------------------|
| Dynix  | 114                 | 92.1                         |
| Trados | 134                 | 85.1                         |
| Total  | 248                 | 88.3                         |

Table 3: Results of running Trados and Dynix data on Alpine

English sentences. This is the prototype, on which users can try their own text. Using this network, the data from the other technical manuals was processed. 248 declarative sentences were extracted. Results are in Table 3, and an example of the output is in the Appendix.

### Limitations

The limits on the length of pre-subject and subject are fairly wide: in the Trados data about 9 out of 134 fell outside these bounds. Other sentences that cannot be processed are those with infinitives as subjects - for instance "To err is human" - and those with relative clauses as subjects - for instance "Whether this is true is debatable". The Appendix gives other examples of failed parses.

However, overall the results are promising. We suggest that this type of system could be used as a pre-processor to facilitate the parsing of longer sentences by other NLP methods.

One of the advantages of this approach to parsing is that it lends itself to the extraction of predicate/argument structure. After the subject has been located the main verb will be found in the predicate, and then the object or complement. With the head of the subject found, we then have the raw material from which we can begin to extract the predicate/argument structure.

### References

- R Duda and P Hart. 1973. *Pattern Classification and Scene Analysis*. John Wiley.
- T Dunning. 1993. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, pages 61–73.
- J L Elman. 1991. Distributed representations, simple recurrent networks and grammatical structure. *Machine Learning*, pages 195–223.
- D Gibbon, R Moore, and R Winski. 1997. *Handbook of Standards and Resources for Spoken Language Systems*. Mouton de Gruyter.
- C L Giles. 1992. Learning and extracting finite state automata with second order recurrent neural networks. *Neural Computation*, pages 393–405.
- S Holden and P Rayner. 1995. Generalization and PAC learning: Some new results for the class of generalized single layer networks. *IEEE Trans. on Neural Networks*, pages 368–380.
- A Lapedes, C Barnes, C Burks, R Farber, and K Sirotkin. 1992. Applications of neural networks and other machine learning algorithms to DNA sequence analysis. In *Computers and DNA*, pages 157–182. Addison Wesley.
- C Lyon and R Frank. 1997. Using Single Layer Networks for Discrete, Sequential Data: an Example from Natural Language Processing. *Neural Computing Applications*, 5 (4).
- Yoh-Han Pao. 1989. *Adaptive pattern recognition and neural networks*. Addison Wesley.
- P. Pym. 1993. Perkins engines and publications. In *PROCEEDINGS of Technology and Language in Europe 2000*. DGXIII-E of the European Commission.
- T Sejnowski and C Rosenberg. 1987. Parallel networks that learn to pronounce English text. *Complex Systems*, pages 145–168.
- C E Shannon. 1951. Prediction and Entropy of Printed English. *Bell System Technical Journal*, pages 50–64.
- R Sutcliffe, H-D Koch, and A McElligott, editors. 1996. *Industrial Parsing of Technical Manuals*. Rodopi.
- L Tarassenko. 1998. *A Guide to Neural Computing Applications*. Wiley.
- P J Wyard and C Nightingale. 1992. A single layer higher order neural net and its application to context free grammar recognition. In N Sharkey, editor, *Connectionist Natural Language Processing*. Intellect.

## Appendix

### Examples from the Trados corpus processed by ALPINE

The prototype locates both the subject and its head.

In Word, { [ you ] } have several possibilities to view non-textual data, such as carriage returns or tabs.

{ Another important [ category ] of non-textual data } is what is referred to as "hidden text."

{ The [ Workbench ] } makes massive use of hidden text in order to perform several tasks .

After opening a new translation unit with the TWB1 or TWB2 buttons,

{ the [ Workbench ] } inserts hidden tags into the Word document .

{ These [ tags ] } delimit the current translation segments, that is, the source segment in the blue source field, and the target segment in the yellow target field as follows .

Later on, { these delimiting [ tags ] } play a crucial role for recognizing "Source Matches" which will be described further down .

After translating, { the source [ segment ] } is kept in your document as "hidden text", together with all delimiting tags .

{ The target [ text ] , that is, the translation you enter, } is of course formatted as normal text, with all formattings intact .

{ A quick [ way ] to toggle between visualizing and hiding nonprinting characters } is again the button in WinWord's standard toolbar .

Examples on which the system will fail include the following.

(i) Test data not well enough modelled by training data

{ A 100% match [ means ] that exactly this sentence } was already translated, and the suggested translation can therefore be accepted as is .

(ii) Idiomatic usage not recognized

{ [ That ] } is, these words make the source sentence longer or shorter than the TM sentence .

(iii) Subject or pre-subject too long

For instance, if a different product name is used in the source sentence than in the fuzzy-match equivalent from TM, both product names will be shown in yellow .