

NEURAL NETWORK DESIGN FOR A NATURAL LANGUAGE PARSER

Caroline Lyon and Ray Frank

Division of Computer Science, University of Hertfordshire, UK *

Abstract

The pattern matching capabilities of neural networks can be mobilised for an automated, natural language, partial parser. First, language complexity is addressed by decomposing the problem into more tractable subtasks. Second, a representation is devised that enables effective, single layer networks to be used to map a pre-defined grammatic framework onto actual sentences. This paper examines data representation, network architecture and learning algorithms appropriate for linguistic data with their characteristic distributions. Users can access a working prototype via telnet on which they can try their own text.

1 Introduction

The natural language parsing task presents a significant challenge, as described in the section headed “The Deplorable State of the Art” of the IBM/Lancaster publication [1]. Our system effectively implements a partial parse, as a foundation for a full parse [2].

Neural networks have been proposed as acceptors of formal languages [3] and our work has a related aim in the natural language field. In training we generate candidate parses for a sentence and use neural nets to accept the desired analysis and reject the others. When used on unseen text over 90% of declarative sentences from technical manuals can be successfully analysed. Typically 10 sentences take about 2 seconds, 50 sentences about 4 seconds to process on a Sparc 10

The texts used in this work come from technical manuals of Perkins Engines Ltd. We first take declarative sentences and decompose them into

pre-subject - subject - predicate

Then each of these constituents can be analysed further. See Figure 1. Using this approach the hierarchical structure of language can be reduced to sets of linear sequences, to be processed independently. For instance, we find the head of the subject in 2 steps, giving the following result:

If a cooler is fitted to the gearbox, { the pipe [connections] of the cooler } must be regularly checked for corrosion. (Sentence 1)

The grammatic framework is not learnt. We assert that a sentence can be decomposed in this way, and then use the neural processors to map the grammar onto each sentence. Some constituents may be empty.

*University of Hertfordshire, Hatfield, Hertfordshire AL10 9AB, UK. Tel: +44 1707 284266 email: C.M.Lyon@herts.ac.uk

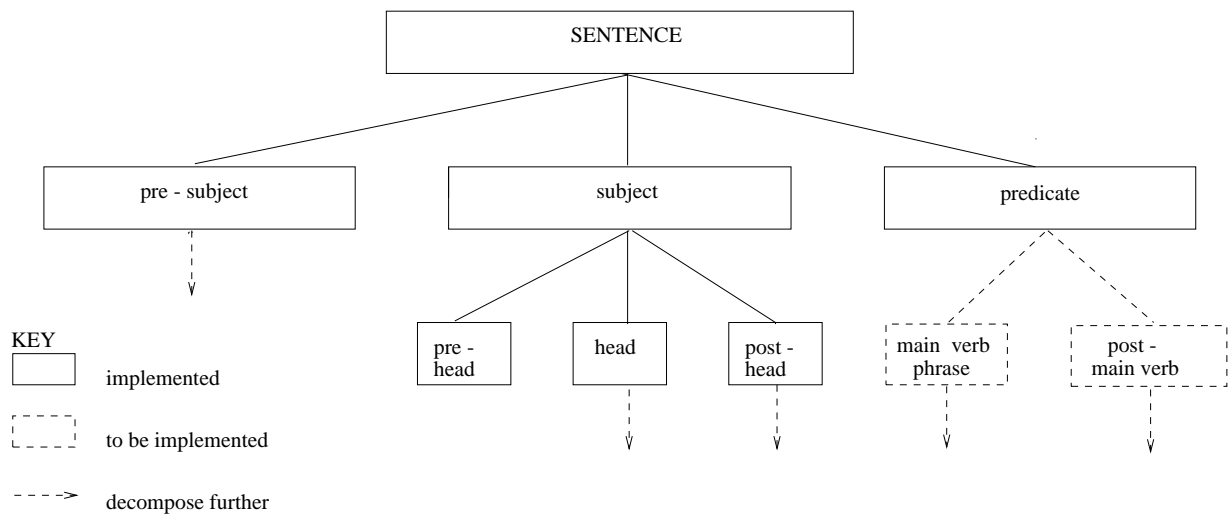


Figure 1: Decomposition of the sentence into syntactic constituents

The advantages of using neural methods

Data driven, neural methods bear comparison with well used stochastic approaches. Neural nets, however, can capture more of the implicit information in the training data since they can model negative as well as positive relationships [2, 4]. Stochastic methods have no obvious way of representing negative relationships. There are negative correlations in natural language that are an important source of information [5, page 80]. We need to distinguish between examples that are never going to be correct and those that just have not occurred in the training set. Neural networks can model both probabilities and “improbabilities”. This is particularly relevant in the natural language field, where data is often found to have a “Zipfian” distribution (see Section 3).

Neural methods also have the advantage that training is done in advance, so the run time computational load is comparatively low.

2 Data Representation

A full description of this process is in [4]. Further details are also found in [2]. The innards of the system can be seen in the experimental prototype accessible via telnet (details from the authors).

Linguistic data is converted to a set of binary input vectors for each sentence, one of which will represent the desired parse, in the following way. First we automatically map words and punctuation marks in a sentence to strings of part-of-speech tags. Customised tagsets, with up to 32 elements, have been developed for each processing stage. Many words can have more than one part-of-speech, and tag disambiguation is a subsidiary task for the first stage of processing. Now, the boundary markers of the syntactic constituents can also be explicitly marked by tags, usually called hypertags and represented by brackets. For the first step the hypertags will demarcate the subject, and strings are generated postulating the placement of hypertags in all possible positions. The hypertags have probabilistic relationships with their neighbours in the same way that adjacent tags do. The neural processor will analyse these relationships and select the string with the desired placement of the hypertags.

Each element of the binary input vector represents an ordered tuple (pair or triple) of adjacent tags, which is flagged to 1 if present in a string. For example, consider the sentence

The light is red. (Sentence 2)

which maps onto:

determiner noun/verb/adjective verb adjective end (2.0)

and would generate strings including:

[determiner] noun verb adjective end (2.1)

[determiner noun] verb adjective end (2.2)

[determiner noun verb] adjective end (2.3)

etc. etc.

String 2.1 would produce the set of tag triples:

([,det,]) (det,],noun) ([,noun,verb) (noun,verb,adj) (verb,adj,end)

The conversion to higher order input serves to capture (partially) the sequential order. This method of representation is supported by analysis of natural language data using information theoretic techniques.

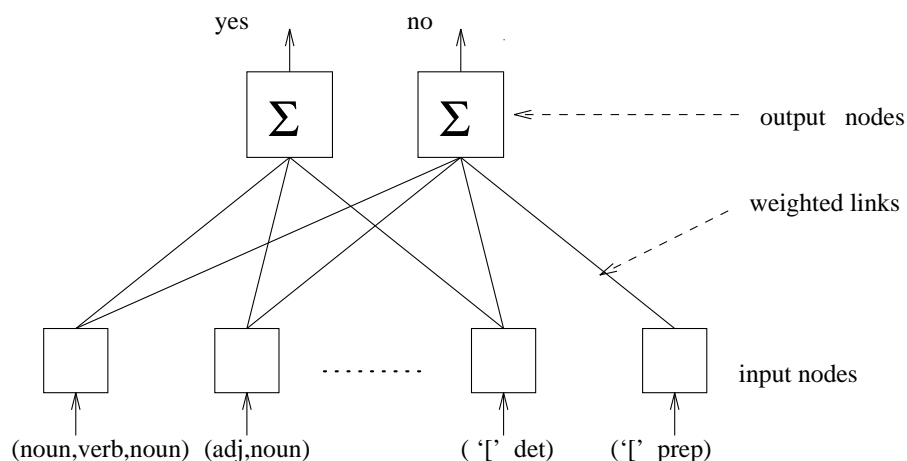
This process produces an unmanageably large number of candidate strings. They are pruned by applying local and semi-local constraints. Local constraints are adjacent tags which are not allowed, such as “determiner - verb” or “start of subject - verb”. If during the generation of a candidate string a prohibited tuple is encountered, then the process is aborted. There are about 100 prohibited pairs and 120 triples in the Prohibition Table. There are also 4 semi-local constraints. For instance, if a relative pronoun occurs, then a verb must follow in that constituent.

The skeletal grammatic framework set out in [4] in EBNF form, also restricts the generation of candidate strings. This grammar is composed of 9 rules - for instance, the subject must contain a noun-type word. Applying this rule to sentence 2 above would eliminate string 2.1. By using these methods the number of candidate strings is drastically reduced. For the technical manuals an average of 4 strings, seldom more than 15 strings, are left. Around 25% of sentences are left with a single string, but the rest can only be parsed using the neural selector.

3 The use of single layer networks

Recent work has lead to a reconsideration of the use of single layer nets. For example, Shavlik et al. compared the performance of single layer perceptron and multi-layer backpropagation nets for 6 processing tests [6, 1992] and concluded “Regardless of the reason, data for many ‘real’ problems seems to consist of linearly separable categories.....”

If the data is not linearly separable, it can sometimes be converted into a linearly separable representation in a higher dimensional space. Widrow [7, 1992] explores this approach which “offers great simplicity and beauty” (page 1420). He examines how a polynomial function can be applied to a set of inputs so that it is transformed into a set of linearly separable data. Yoh-Han Pao (1989) takes this view further, and examines how input data can be enhanced so that supervised learning problems can be solved with a single layer feed forward network [8]. For the work described here we find empirically that with data pre-processed into higher order inputs it is linearly separable. Running on a traditional perceptron data can be trained to about 99%.



'[' 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. Σ represents summing function.

Figure 2: The single layer net, not fully connected

The learning algorithm

Having established that the data is linearly separable, and hence that single layer networks can be used, further choices on network function arise. First, consider a multi-layer network where the classic problem of penalty assignment must be addressed. It is necessary to construct an error function which measures the network's performance as a differentiable function of the trainable weights. The well known procedure that is employed in backpropagation utilises a method based on minimising the error between the desired and actual activation of the output nodes. This may be called an *internal* error measure, and weight changes on connections are related to it.

Now, if we have a single layer network different functions can be used to assign penalties and trigger weight updates on connections. Since the net has no hidden layer, the blame for an incorrect output can be directly assigned to the inputs contributing to it. We may still use a method of minimising the internal error. An alternative mechanism, employed here, is to trigger a weight update from a misclassification, which is not necessarily the same [9]. This can be called an *external* error measure. The weights can be updated directly, the change need not be based on the internal error level.

In our earlier work, the performance of a single layer network functioning with an external error metric was compared to that of a backpropagation net. The back propagation net performed slightly less well [4] and we looked for an explanation.

One possibility is that the type of single layer network used here is more appropriate for processing characteristic linguistic data. Since the update factor depends on the current value of the weights, not the error measure, the adjustment for a small error can be as great as for a large error: a miss is as good as a mile. This seems to be an appropriate method in grammaticality detection, since we aim to make absolute distinctions between correct and incorrect strings. Many of the strings that should be rejected differ in a single tag from the correct version. We need to classify

not only grossly ungrammatical sentences, but also these near misses.

The type of data distribution is also relevant. Rare examples, rather than being noise, can contribute to the classification task. Shannon's classic work cited Zipf's law on the distribution of words [10]. Informally stated, this law captures the fact that a small number of words are very common, but a significant number are used infrequently. Zipfian distribution of other linguistic elements has been marked, and it characterises part-of-speech tag tuples. It is necessary to get **some** information from relatively infrequent as well as common events, and then exploit the redundancy of natural language. We want a single contrary example to carry its due weight. Using a method in which misclassifications directly trigger weight adjustments helps to achieve this.

The net used here is derived from Wyard and Nightingale's Hodyne net [11]. It is chosen because of its simplicity and effectiveness. The nets are trained in supervised mode on marked up training examples. Typically, 300 sentences generate about 1200 strings which train to over 99% correct in about 2 seconds. Initially the weighted links are disabled. When a string is presented to the network in training mode, it activates a set of input nodes. If an input node is not already linked to the output node representing the desired response, it will be connected and the weight on the connection will be initialised to 1.0. Some nodes, a minority, will only be connected to one output - see Figure 2. The input layer potentially has a node for each possible tuple, $32^3 + 32^2$ for pairs and triples. In practice there is an upper bound of 1000 on the number activated. (In testing mode, if a previously unseen tuple appears it makes zero contribution to the result.) The activations at the input layer are fed forward through the weighted connections to the output nodes, where they are summed. The highest output marks the winning node. If the desired node wins, then no action is taken. If the desired node does not win, then the weight on connections to the desired node are incremented, while the weights on connections to the unwanted node are decremented.

We are currently using the original Hodyne function. If $\delta = +1$ for strengthening weights and $\delta = -1$ for weakening them, then

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

The update factor should always be positive, and asymptotic to maximum and minimum bounds. It should be greatest in the central region, least as it moves away in either direction. These requirements are satisfied.

4 Results

When the system is used to analyse unseen text the output is interpreted in the following way. The difference between the "yes" and "no" activation levels is recorded for each string, and this score is considered a measure of grammaticality, Γ . The string with the highest Γ score is taken as the correct one. Table 1 gives a summary of results, showing how successfully the subject and head were located. The metric "with tags right" indicates how many sentences have all words correctly tagged too. For these results the networks were trained on one part of the corpus of 351 declarative sentences and tested on another part. There are currently arbitrary limits on the length of the pre-subject (15 words) and subject (12 words), which means that 2% of the sentences cannot be processed. These limitations are less restrictive than those usually encountered [1]. The average sentence length is 18 words. Further information on training and test data, and on performance is given in [12].

no. of training sents.	no. of test sents.	% sents with subject found	% sents with tags right	% sents with subject and head found
309	42	100	97.6	97.6
288	63	98.4	96.8	96.8
292	59	98.3	98.3	96.6
284	67	94.0	94.0	94.0

Table 1: Performance on declarative sentences from Perkins manuals, after 2% have been excluded

The experimental prototype on which users can process their own text was trained on the whole corpus from the technical manuals, slightly augmented. It is accessible through telnet (details from the authors). This demonstrates how neural technology can be mobilised for the natural language parsing task.

References

- [1] E Black, R Garside, and G Leech. *Statistically driven computer grammars of English: the IBM/Lancaster approach*. Rodopi, 1993.
- [2] C Lyon and R Dickerson. A fast partial parse of natural language sentences using a connectionist method. In *7th Conf. of European Chapter of Association of Computational Linguistics*, 1995.
- [3] H T Siegelmann, E D Sontag, and C Lee Giles. Complexity of language recognition by neural networks. In *12th World Computer Congress on Algorithms, Software and Architecture*. Elsevier, 1992.
- [4] C Lyon. *The representation of natural language to enable neural networks to detect syntactic structures*. PhD thesis, University of Hertfordshire, 1994.
- [5] E Charniak. *Statistical Language Learning*. MIT, 1993.
- [6] J Shavlik, R Mooney, and G Towell. Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 1992.
- [7] B Widrow and M Lehr. 30 years of adaptive neural networks. In C Lau, editor, *Neural Networks: theoretical foundations and analysis*. IEEE press, 1992.
- [8] Yoh-Han Pao. *Adaptive pattern recognition and neural networks*. Addison Wesley, 1989.
- [9] M L Brady, R Raghavan, and J Slawny. Back propagation fails to separate where perceptrons succeed. *IEEE Trans. on Circuits and Systems*, 1989.
- [10] C E Shannon. Prediction and Entropy of Printed English. *Bell System Technical Journal*, 1951.
- [11] 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.
- [12] C Lyon. Guidelines for neural network design and their application to natural language processing. Technical report, School of Information Sciences, University of Hertfordshire, September 1995.