Learning Incremental Syntactic Structures with Recursive Neural Networks

F. Costa¹, P. Frasconi¹, V. Lombardo², and G. Soda¹
¹Dept. of Systems and Computer Science, University of Florence
via di Santa Marta 3, I-50139 Firenze (Italy)
²DISTA, Università del Piemonte Orientale “A. Avogadro”
c.so Borsalino 54, I-15100 Alessandria (Italy)

Abstract
We develop novel algorithmic ideas for building a natural language parser grounded upon the hypothesis of incrementality, which is widely supported by experimental data as a model of human parsing. Our proposal relies on a machine learning technique for predicting the correctness of partial syntactic structures that are built during the parsing process. A recursive neural network architecture is employed for computing predictions after a training phase on examples drawn from a corpus of parsed sentences, the Penn Treebank. Our results indicate the viability of the approach and lay out the premises for a novel generation of algorithms for natural language processing which more closely model human parsing. These algorithms may prove very useful in the development of efficient parsers and have an immediate application in the construction of semiautomatic annotation tools.

1 Introduction
Incremental processing of the human parser is a widely held assumption in psycholinguistics. This assumption accounts for the intuitive fact that language is processed from left to right (like it arrives at human ears as speech), and the experimentally proven fact that humans interpret language well before reaching the end of the input sentence, that is they are able to assign a meaning to "almost any" initial (left) fragment of a sentence (Marslen-Wilson, 1973).

Although incremental processing of natural language is uncontroversially supported by many psycholinguistic experiments, effective models for parsing unconstrained language are rarely based on the incrementality hypothesis. In this area, most solutions consist in stochastic versions of bottom-up parsers, relying on models grounded upon context-free rule probabilities (Collins, 1996). The incremental approach is at work only in psycholinguistic models of the human parser, mostly tested on artificially-generated sentences that display this or that ambiguity form. Probably, the major reason is the incomplete knowledge about the human parser, which makes a reliable algorithmic design impossible.

However, some of the discovered features of the human parser have already been exploited in the NLP system implementation. For example, a number of preference heuristics (like Late Closure (Kimball, 1973) or Minimal Attachment (Frazier & Fodor, 1978)), devised in psycholinguistic environments to describe some aspects of the human parser, have been implemented to solve local ambiguities (see, e.g., (Hobbs & Bear, 1990)), and, after the emergence of the corpus-based approaches, NLP systems often involve a combination of heuristics and statistics to guide the parser decisions. In this paper, we adopt the novel assumption that the overall parsing strategy incorporates a major issue of the human parser, namely its incremental nature.

An operational account of the incrementality hypothesis, called strong incrementality, is at the core of a number of computational models of the human parser (Milward, 1995; Stabler, 1994), as a parsimonious algorithmic version of incrementality. In this version, the parser maintains a totally connected parse, while scanning the input words from left to right, with no input stored in a disconnected state. The major difficulty of implementing a strongly incremental strategy is that the connection of the next input word to the existing structure (what is called the left context requires the construction of a substructure called the connection path (CP) see Figure 1). This results in a high number of candidate CPs, which yields a hard search problem, affecting the parser accuracy and speed.

In this paper, we propose an incremental parsing model able to process unconstrained naturally occurring texts. At each step of parsing, the correct CP linking the next word to the incrementally built parse tree is predicted using a neural network model. Such a prediction effectively acts like a heuristic that can guide the search process, and significantly reduce computational efforts. Designing accurate heuristics for this problem is a difficult task but, on the other hand, availability of databases of parsed sentences (treebanks) makes the
problem interesting from a machine learning perspective. In particular, we are interested in a solution based on connectionist models.

Each instance in the learning domain is naturally represented as a syntactic structure, whose nodes are labeled by linguistic symbols. Each structure is obtained by joining a candidate CP to a given incremental tree. The prediction task consists of estimating the probability that a given attachment is correct. Examples are drawn from a subset of parsed sentences in the Penn Treebank (Marcus et al., 1993). The structured nature of instances in the learning domain makes this task suitable for the application of recursive neural networks (also known as folding architecture), that can solve the supervised learning problem on structured data (Goller & Küchler, 1996; Sperduti & Starita, 1997; Frasconi et al., 1998). This model has been already proved to be effective on another related symbolic task aimed at learning search heuristics for guiding a theorem prover (Goller, 1997).

2 The incremental parsing algorithm

The incrementality hypothesis poses some constraints upon the general parsing strategy. In its general form, the hypothesis implies that the semantic interpretation of some fragment of the sentence is available as the scan of the input material proceeds from left to right. This does not involve any specific constraint on how the interaction between the syntactic analysis and the semantic interpretation can be implemented at the algorithmic level.

One possible (and parsimonious) solution to the interaction problem is offered by the strongly incremental version of the hypothesis. The interaction occurs at the word level, and compels the syntactic analyzer to keep a totally connected structure at all times (that is, the semantic interpreter has no means to work on disconnected items). To be effective, the implementation of

a strongly incremental parser must reduce at its best all the failures due to disambiguation procedures about the attachment of a word to the current left context. The key concept in attaching a word to the left context is the notion of connection path (CP) (Lombardo & Sturt, 1999), which is the chain of syntactic nodes that must be built in order to appropriately connect the current word to the left context. The strongly incremental parsing algorithm proceeds from left to right, and for each word w, computes the CPs that allow to link it with some node on the right edge of the left context. A selection procedure chooses the best CP for continuation, and instantiates it to generate the new incremental tree. The better is the selection procedure, the less parsing decisions are wrong. The major point of this paper is to apply the connectionist architecture described in the next section to solve the local ambiguity due to the selection of the connection path.

In order to evaluate the applicability of such a methodology, we have designed an experiment which consists in the simulation of an incremental parsing algorithm on the trees of the Penn Treebank. The simulation algorithm, which is described in detail in (Lombardo & Sturt, 1999), computes for each word in the input the CP required to connect it to the left context. The training phase of the connectionist architecture takes into account both the positive and the negative data; that is, given a word it takes into account all the CPs in the data base that are applicable (included the correct one). In the test phase, the network takes in input all the CPs that are applicable in some situation (or, more precisely, the trees that result from the application of the appropriate connection paths), and returns a ranking of the alternatives.

In section 4, after the description of the network architecture, there is a discussion of the results we have yielded on a sample from the Penn Treebank.

3 Neural network architecture

Here we give a short description of the recursive neural network model, specialized for the case of labeled ordered trees. More details about the architecture and learning algorithms are described in (Frasconi et al., 1998). The supervised learning task is formalized as the problem of estimating the conditional probability P(O|I). Here O is a binary output variable associated with the class and I is a labeled ordered m-ary tree (as syntactic trees are). By ordered we mean that for each vertex v a total order is defined on the m children of v. Specifically, ch[v] denotes the ordered m-tuple of vertices whose elements are v's children; if v has k < m children, then the last m – k elements of ch[v] are filled with a special entry nil denoting a missing child. I(v) is the label attached to vertex v. In this paper, labels are syntactic categories, modeled as random variables
with realizations in a finite alphabet \( I = I_1, \ldots, I_N \).

\[
x(v) = f(x(ch[v]), I(v))
\]

\[
o = g(x(r)).
\]

Figure 2: The recursive neural network model.

The basic neural network architecture is based on the following recursive state space representation:

\[
x(v) = f(x(ch[v]), I(v))
\]

\[
o = g(x(r)).
\]

In the above equation, \( x(v) \in R^n \) denotes the state vector associated with node \( v \) and \( x(ch[v]) \in R^{m \times n} \) is a vector obtained by concatenating the components of the state vectors contained in \( v \)'s children. \( f : R^{m \times n} \times I \to R^n \) is the state transition function that maps states at \( v \)'s children and the label at \( v \) into the state vector at \( v \). \( g : R^n \to [0, 1] \) is the output function that maps the state \( x(r) \) at the root of the tree into a real number that can be interpreted as \( P(O = 1|I) \), the probability that the input tree \( I \) is a positive example. States in Eq. (1) are updated bottom-up, traversing the tree in post-order. If a child is missing, the corresponding entries in \( x(ch[v]) \) are filled with the frontier state \( \overline{X} \) which is associated with the base step of recursion.

The transition function \( f \) is implemented by a feedforward neural network according to the following scheme (see Figure 2):

\[
a_i(v) = \omega_i \circ \sum_{j=1}^{N} \omega_{i,j} z_j(I(v)) + \sum_{k=1}^{m} \sum_{l=1}^{n} w_{k,l} x_k(ch[v])
\]

\[
x_i(v) = \tanh(a_i(v)) \quad i = 1, \ldots, n
\]

where \( z_j(v) \) denotes the \( j \)-th component of the state vector at vertex \( v \), \( z_j(I_0) = 1 \) if \( j = q \) or zero otherwise (i.e., we are using a one-hot encoding of symbols), \( ch_k[v] \) denotes the \( k \)-th child of \( v \), and \( \omega_{i,j}, w_{k,l} \) are adjustable weights. The output function \( g \) is implemented by another network:

\[
o = \sigma \left( w_0 \sum_{j=1}^{n} w_j x_j(r) \right)
\]

being \( \sigma \) the logistic function and \( w_j \) adjustable weights.

After the recursion (1) is completed, the state vector \( x(r) \) at the root \( r \) contains an encoding of the whole tree. This is similar to the encoding performed by recursive autoassociative memories first proposed by Pollack (Pollack, 1990) but here the encoding results from a supervised learning mechanism. Supervised learning follows the maximum likelihood principle as in many other neural network models. Optimization is solved by gradient descent, with gradients computed by the backpropagation through structure algorithm (Goller & Kühn, 1996).

4 Implementation and results

The whole dataset available for the experiments was built with 2000 parsed sentences extracted from the Wall Street Journal Section of the Penn II Treebank (Marcus et al., 1993), as follows. First, we extract a list \( L \) of 1675 CPs from the incrementally pre-processed annotated corpus of 2000 sentences, as described in (Lombardo & Sturt, 1999). Then, for each sentence and for each incremental tree \( T_i \) within the sentence we extract all the valid CPs drawing them from \( L \). A CP is valid for \( T_i \) if its anchor and its foot can match the corresponding entities in \( T_i \). In order to search valid CPs, we first scan the right frontier of \( T_i \) (any of these nodes may be the anchor) and then \( L \) is searched for CPs matching the anchor and the foot. For each \( T_i \), we obtain in this way a candidate list of CPs, denoted \( C_i \). On this corpus the average length of \( C_i \) is 57 while the maximum length is 387. The resulting incremental trees are then constructed by joining each CP in \( C_i \) with \( T_i \) to form a candidate list \( T_{i+1} \) of incremental trees. For each \( T_i \) there is only one valid incremental tree \( T_{i+1} \) (as determined by the parsed sentence in the treebank) in \( T_{i+1} \). This tree is marked as a positive example while all the remaining trees in \( T_{i+1} \) are marked as negative examples. All this has as a consequence that every positive example has to stand against 57 negative examples on average. In order not to induce the learning algorithm to constantly classify as negative every instance, negative examples have been presented in the training phase with a lesser probability than the positive ones so to reestablish a ratio of 1:1 on average for each epoch but ensuring (in a probabilistic sense) to present every negative example given a sufficiently high number of epochs. Candidate incremental trees are fed into the RNN model by discarding leaves (labeled by words in the lexicon) and using 71 distinct grammatical categories as node labels.

The recursive network used in the experiments implements the model described in Section 3 and has the following specifications: \( m = 14 \) (maximum outdegree), \( N = 71 \) labels (one-hot encoded), and \( n = 60 \) state components, yielding total of about 54,000 adjustable parameters. In the training phase, a set of 100 sentences has been used for a total of 183,000 incremental trees (2,685 of these were positive examples), and other nonoverlapping set of 100 different sentences has been used as a validation set for early stopping of gradient descent. A third set of nonoverlapping 500 sentences
has been reserved for testing generalization.

After training, the network assigns a correctness probability to each candidate CP. In a real parser or in a semi-automatic annotation tool (Lombardo et al., 2000), the network output can be employed to sort candidates by increasing probability of correctness. Hence, our measure of performance is based on the rank assigned to the correct CP in each candidate list $T_i$. A synthesis of results on the 500 test sentences is reported in Table 1. For each candidate list length $|T_i|$ we measured the average position $R$ of the correct CP after sorting the list according to network predictions. $R = 1$ would yield a perfect predictor that always ranks the correct CP in first position. For brevity, results are averaged over 11 bins associated different candidate list lengths. The total number of candidates in the lists of each bin is reported in the last column of Table 1. Although preliminary, these results are very encouraging and clearly demonstrate that the ranking suggested by the network significantly reduces the number of candidate CPs at each incremental parsing step. These predictions can be very effectively integrated in a semi-automatic annotation tool. The software tool parses the sentence from left to right, and interacts with the human annotator in case of ambiguity. The tool displays the possible alternatives as ranked according to the network result, and the annotator chooses the correct one. The earlier appears the correct choice in the presentation order, the faster is the parsing process. For all the 500 test sentences, the total number of incremental trees is 10,232 and more than 90% of the times the number of alternative CPs is between 2 and 150 (see Table 1). In all these 90% of cases, the correct CP is suggested by our predictor within the first 9 positions, on average. Similarly, 96% of the times the number of alternatives is between 2 and 200 and in these cases the correct CP can be found within the first 12 positions on average. On the same 500 test sentences, we have also computed that presenting to the human annotator only the 5 best alternatives we have a probability of 81.4% to find the correct CP listed.

| $|T_i|$ | $R$ (± std. dev.) | # candidates |
|------|-----------------|--------------|
| 2-5  | 1.51 ± 0.33     | 753          |
| 6-10 | 2.20 ± 1.63     | 1,320        |
| 11-20| 2.50 ± 2.27     | 1,512        |
| 21-30| 3.97 ± 4.67     | 1,126        |
| 31-40| 4.36 ± 5.22     | 1,010        |
| 41-50| 4.09 ± 6.04     | 876          |
| 51-70| 5.29 ± 8.51     | 1,433        |
| 71-100| 6.98 ± 12.52   | 1,497        |
| 101-150| 8.87 ± 17.78  | 1,206        |
| 151-200| 11.97 ± 26.96 | 578          |
| 201-387| 17.90 ± 40.77 | 376          |

| Table 1: Prediction performance (see text for explanation). |

5 Conclusions

The application of such results to parsing are immediate: a long term project consists in the development of an efficient incremental parser which is informed by the network predictions in taking decisions about attachment ambiguity; a short term project consists in the development of the interactive tool described above, which is of support to human treebank annotators (Lombardo et al., 2000). The methodology illustrated in this paper uniquely relies upon algorithmic issues related to the incrementality hypothesis. A possible empirical augmentation of this methodology, that can make it better viable for totally automatic envi-

ronments, is the introduction of a number of linguistic accounts, that could provide a theoretically-grounded reduction of the candidate CPs.

References