In Proc. of 10th Conf. of European Chapter of the Association for Computational Linguistics (EACL 2003).

Neural Network Probability Estimation for Broad Coverage Parsing

James Henderson
Département d'Informatique
Université de Genève
James.Henderson@cui.unige.ch

Abstract

We present a neural-network-based statistical parser, trained and tested on the Penn Treebank. The neural network is used to estimate the parameters of a generative model of left-corner parsing, and these parameters are used to search for the most probable parse. The parser's performance (88.8% F-measure) is within 1% of the best current parsers for this task, despite using a small vocabulary size (512 inputs). Crucial to this success is the neural network architecture's ability to induce a finite representation of the unbounded parse history, and the biasing of this induction in a linguistically appropriate way.

1 Introduction

Many statistical parsers (Ratnaparkhi, 1999; Collins, 1999; Charniak, 2001) are based on a history-based probability model (Black et al., 1993), where the probability of each decision in a parse is conditioned on the previous decisions in the parse. A major challenge in this approach is choosing a representation of the parse history from which the probability for the next parser decision can be accurately estimated. Previous approaches have used a hand-crafted finite set of features to represent the unbounded parse history (Ratnaparkhi, 1999; Collins, 1999; Charniak, 2001). In the work presented here, we automatically induce a finite set of features to represent the unbounded parse history. We perform this induction using an artificial neural network architecture, called Simple Synchrony Networks (SSNs) (Lane and Henderson, 2001; Henderson, 2000). Because this architecture is specifically designed for processing structures, it allows us to impose structurally specified and linguistically appropriate biases on the search for a good history representation. The resulting parser achieves performance far greater than previous approaches to neural network parsing (Ho and Chan, 1999; Costa et al., 2001), and only marginally below the current state-of-the-art for parsing the Penn Treebank.

We propose a hybrid parsing system consisting of two components, a neural network which estimates the parameters of a probability model for phrase structure trees, and a statistical parser which searches for the most probable phrase structure tree given these parameters. We first present the probability model which is common to these two components, followed by the estimation method, the search method, and a discussion of the empirical results.

2 The Generative Probability Model

The probability model we use is generative and history-based. Generative models are expressed in terms of a stochastic process which generates both the phrase structure tree and the input sentence. At each step, the process chooses a characteristic of the tree or predicts a word in the sentence. This sequence of decisions is the derivation of the tree, which we will denote $d_1,..., d_m$. Because there is a one-to-one mapping from phrase structure trees to our derivations, the probability of a derivation $P(d_1,..., d_m)$ is equal to the joint probability of the derivation's tree and the input sentence. The probability of the input sentence is a constant across all the candidate derivations, so we only need to find the most probable derivation. In history-based models (Black et al., 1993), the probability estimate for each derivation decision d_i is conditioned on the previous derivation decisions $d_1,..., d_{i-1}$, which is called the derivation history at step i. This allows us to use the chain rule for conditional probabilities to derive the probability of the entire derivation as the multiplication of the probabilities for each of its decisions.

$$P(d_1,..., d_m) \;\; = \;\; \Pi_i P(d_i|d_1,..., d_{i-1})$$

The probabilities $P(d_i|d_1,..., d_{i-1})$[1] are the parameters of the parser's probability model.

To define the parameters $P(d_i|d_1,..., d_{i-1})$ we need to choose the ordering of the decisions in a derivation, such as a top-down or shift-reduce ordering. The ordering which we use here is that of a form of left-corner parser (Rosenkrantz and Lewis, 1970). A left-corner parser decides to introduced a node into the parse tree after the subtree rooted at the node's first child has been fully parsed. Then the subtrees for the node's remaining children are parsed in their left-to-right order. We use the binarized version of a left-corner parser, described in (Manning and Carpenter, 1997), where the parse of each non-leftmost child begins with the parent node predicting the child's leftmost terminal, and ends with the child's root nonterminal attaching to the parent. An example of this ordering is shown by the numbering on the left in figure 1. The process which generates a tree begins with a stack that contains a node labeled *ROOT* (step 0) and must end in the same configuration (step 9), as shown on the right of the figure. The possible derivation decisions are: predict the next tag-word pair and push it on the stack (steps 1, 4, and 6), replace the node on top of the stack with a new node which is its parent and choose the label of that node (steps 2, 3, and 5), and pop a node from the stack and attach it as the child of the node below it on the stack (steps 7, 8, and 9).[2]

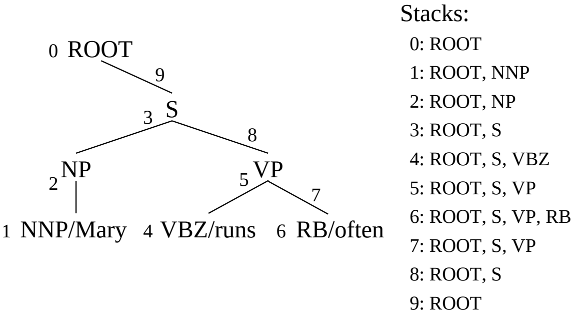

Figure 1: The decomposition of a parse tree into derivation decisions (left) and the stack after each decision (right).

3 Inducing Features of the Derivation History

The most important step in designing a statistical parser with a history-based probability model is choosing a method for estimating the parameters $P(d_i|d_1,..., d_{i-1})$. The main difficulty with this estimation is that the history $d_1,..., d_{i-1}$ is of unbounded length. Most probability estimation methods require that there be a finite set of features on which the probability is conditioned. The standard way to handle this problem is to handcraft a finite set of features which provides a sufficient summary of the unbounded history (Ratnaparkhi, 1999; Collins, 1999; Charniak, 2000). The probabilities are then assumed to be independent of all the information about the history which is not captured by the chosen features. The difficulty with this approach is that the choice of features can have a large impact on the performance of the system, but it is not feasible to search the space of possible feature sets by hand. One alternative to choosing a finite set of features is to use kernel methods, which can handle unbounded

[1] When $i = 1$, $P(d_i|d_1,..., d_{i-1}) = P(d_1)$.

[2] We extended the left-corner parsing model in a few minor ways using grammar transforms. We replace Chomsky adjunction structures (i.e. structures of the form [X [X ...] [Y ...]]) with a special "modifier" link in the tree (becoming [X ... [$_{mod}$ Y ...]]), requiring nodes which are popped from the stack to choose between attaching with a normal link or a modifier link. We also compiled some frequent chains of non-branching nodes (such as [S [VP ...]]) into a single node with a new label (becoming [S-VP ...]). These transforms are undone before any evaluation is performed on the output trees. We do not believe these transforms have a major impact on performance, but we have not currently run tests without them.

feature sets, but then efficiency becomes a problem. Collins and Duffy (2002) define a kernel over parse trees and apply it to re-ranking the output of a parser, but the resulting feature space is restricted by the need to compute the kernel efficiently, and the results are not as good as Collins' previous work on re-ranking using a finite set of features (Collins, 2000).

In this work we use a method for automatically inducing a finite set of features for representing the derivation history. The method is a form of multi-layered artificial neural network called Simple Synchrony Networks (Lane and Henderson, 2001; Henderson, 2000). The outputs of this network are probability estimates computed with a log-linear model (also known as a maximum entropy model), as is done in (Ratnaparkhi, 1999). Log-linear models have proved successful in a wide variety of applications, and are the inspiration behind one of the best current statistical parsers (Charniak, 2000). The difference from previous approaches is in the nature of the input to the log-linear model. We do not use handcrafted features, but instead we use a finite vector of real-valued features which are induced as part of the neural network training process. These induced features represent the information about the derivation history which the training process has decided is relevant to estimating the output probabilities. In neural networks these feature vectors are called the hidden layer activations, but for continuity with the previous discussion we will refer to them as the history features.

We will denote the history feature computation with the function h, and the output log-linear model with the function o, whose result is a probability distribution over the possible derivation operations.

$$o_{d_i}(h(d_1,...,d_{i-1})) \approx P(d_i|d_1,...,d_{i-1})$$

The mapping h from the derivation history to the history features is computed with the repeated application of a function g, which maps previous history representations plus pre-defined features of the derivation history to a real-valued vector.

Because the function g is nonlinear, the induction of these features allows the training process to explore a much more general set of estimators $o(h(x))$ than would be possible with a log-linear model alone (i.e. $o(x)$).[3] This generality makes this estimation method less dependent on the choice of input representation x. In addition, because the inputs to g include previous history representations, the mapping h is defined recursively. This recursion allows the input to the system to be unbounded, thereby allowing an unbounded derivation history to be successively compressed into a fixed-length vector of history features.

Training a Simple Synchrony Network (SSN) is similar to training a log-linear model. First an appropriate error function is defined for the network's outputs, and then some form of gradient descent learning is applied to search for a minimum of this error function.[4] This learning simultaneously tries to optimize the parameters of the output computation o and the parameters of the mapping h from the derivation history to the history features. With multi-layered networks such as SSNs, this training is not guaranteed to converge to a global optimum, but in practice a set of parameters whose error is close to the optimum can be found. The reason no global optimum can be found is that it is intractable to find the optimal mapping h from the derivation history to the history features. Given this difficulty, it is important to impose appropriate biases on the search for a good set of history features.

The main bias we have exploited in this work is the recency bias in training recursively defined neural networks. The only trained parameters of the mapping h are the parameters of the function g, which records a subset of the information from a set of previous history representations in a new history representation. The training process auto-

[3] As is standard, g is the sigmoid activation function applied to a weighted sum of its inputs. Multi-layered neural networks of this form can approximate arbitrary mappings from inputs to outputs (Hornik et al., 1989), whereas a log-linear model alone can only estimate probabilities where the category-conditioned probability distributions $P(x|d_i)$ of the pre-defined inputs x are in a restricted form of the exponential family (Bishop, 1995).

[4] We use the cross-entropy error function, which ensures that the minimum of the error function converges to the desired probabilities as the amount of training data increases (Bishop, 1995). This implies that the minimum for any given dataset is an estimate of the true probabilities. We use the on-line version of Backpropagation to perform the gradient descent.

matically chooses these parameters based on what information needs to be recorded. The recorded information may be needed to compute the output for the current step, or it may need to be passed on to future history representations to compute a future output. However, the more history representations intervene between the place where the information is input and the place where the information is needed, the less likely the training is to learn to record this information. We can exploit this recency bias in inducing history representations by ensuring that information which is known to be important at a given step in the derivation is input directly to that step's history representation, and that as information becomes less relevant it has increasing numbers of history representations to pass through before reaching the step's history representation. In the next section we will present how this inductive bias is exploited in the design of the SSN parser.

4 Estimating Derivation Probabilities with a Simple Synchrony Network

Simple Synchrony Networks are an artificial neural network architecture which is specifically designed for processing structured data. A SSN divides the processing of a structure into a set of sub-processes, with one sub-process for each node of the structure. For phrase structure tree derivations, we divide a derivation into a set of sub-derivations by assigning a derivation step i to the sub-derivation for the node top_i which is on the top of the stack prior to that step. The SSN network then performs the same computation at each position in each sub-derivation. The unbounded nature of phrase structure trees does not pose a problem for this approach, because increasing the number of nodes only increases the number of times the SSN network needs to perform a computation, and not the number of parameters in the computation which need to be trained.

Each computation which the network performs results in two real-valued vectors, namely the results of the functions o and h. As discussed in the previous section, the function o is simply a log-linear model applied to the result of h. When h is applied to node top_i at step i, it computes the history representation $h(d_1,..., d_{i-1})$ by applying the function g to a set of pre-defined features $f(d_1,..., d_{i-1})$ of the derivation history plus a small set of previous history representations.

$$h(d_1,..., d_{i-1}) = g(f(d_1,..., d_{i-1}), \{rep_{i-1}(c) | c \in D(top_i)\})$$

where $rep_{i-1}(c)$ is the most recent previous history representation for a node c.

$$rep_j(c) = h(d_1,..., d_{\max(k|k \leq j \wedge top_k = c)})$$

$D(top_i)$ is a small set of nodes which are in a structurally local domain of top_i. This domain always includes top_i itself, but the remaining nodes in $D(top_i)$ and the features in $f(d_1,..., d_{i-1})$ need to be chosen by the system designer. These choices determine how information flows from one set of history features to another, and thus determines the inductive bias discussed in the previous section.

The principle we apply when designing $D(top_i)$ and $f(d_1,..., d_{i-1})$ is that we want the inductive bias to reflect structural locality. For this reason, $D(top_i)$ includes nodes which are structurally local to top_i. These nodes are the left-corner ancestor of top_i (which is below top_i on the stack), top_i's left-corner child (its leftmost child, if any), and top_i's most recent child (which was top_{i-1}, if any). For right-branching structures, the left-corner ancestor is the parent, conditioning on which has been found to be beneficial (Johnson, 1998), as has conditioning on the left-corner child (Roark and Johnson, 1999). Because these inputs include the history features of both the left-corner ancestor and the most recent child, a derivation step i always has access to the history features from the previous derivation step $i - 1$, and thus (by induction) any information from the entire previous derivation history could in principle be stored in the history features. Thus this model is making no a priori hard independence assumptions, just a priori soft biases.

As mentioned above, $D(top_i)$ also includes top_i itself, which means that the inputs to g always include the history features for the most recent derivation step assigned to top_i. This input imposes an appropriate bias because the induced history features which are relevant to previous derivation decisions involving top_i are likely to be relevant to the decision at step i as well. As a simple example, in figure 1, the prediction of the left corner terminal of the *VP* node (step 4) and the decision that the *S* node is the root of the whole sentence (step 9) are both dependent on the fact that the node on the top of the stack in each case has the label *S* (chosen in step 3).

The pre-defined features of the derivation history $f(d_1,...,d_{i-1})$ which are input to g for node top_i at step i are chosen to reflect the information which is directly relevant to choosing the next decision d_i. In the parser presented here, these inputs are the last decision d_{i-1} in the derivation, the label or tag of the sub-derivation's node top_i, the tag-word pair for the most recently predicted terminal, and the tag-word pair for top_i's left-corner terminal (the leftmost terminal it dominates). Inputting the last decision d_{i-1} is sufficient to provide the SSN with a complete specification of the derivation history. The remaining features were chosen so that the inductive bias would emphasize these pieces of information.

5 Searching for the Best Parse

Once we have trained the SSN to estimate the parameters of our probability model, we use these estimates to search the space of possible derivations to try to find the most probable one. Searching the space of all possible derivations has exponential complexity, so it is important to be able to prune the search space. Being able to prune effectively is particularly important for neural network approaches, due to the computational cost of computing probability estimates. We use a form of beam search to prune the search space.

The choice of the left-corner ordering for derivations is crucial to the success of this neural network parser in that it allows very severe pruning without significant loss in performance. The most important pruning occurs after each word has been predicted and pushed on the stack (for example, after steps 1, 4, and 6 in figure 1). When a partial derivation reaches this position it is stopped to see if it is one of a small number of the best partial derivations which end in predicting that word. The search only pursues a beam of the best 100 derivations past each word prediction. Experiments with a variety of beam widths confirms that little if any validation performance is gained with larger beam widths. To search the space of derivations in between two word predictions we do a best-first search. This search is not restricted by a beam width, but a limit is placed on the search's branching factor. At each point in a partial derivation which is being pursued by the search, only the 10 best alternative decisions are considered for continuing that derivation. This was done because we found that the best-first search tended to pursue a large number of alternative labels for a nonterminal before pursuing subsequent derivation steps, even though only the most probable labels ended up being used in the best derivations. We found that a branching factor of 10 was large enough that it had virtually no effect on validation performance.

The most computationally intensive operation of the parser is computing the probability estimates for the predictions of the next tag-word pair. To compute the log-linear model of this prediction, it is necessary to compute values for all possible next words, not just the correct next word, because they are needed for normalizing. Because there are a very large number of words, this is expensive. To reduce this burden, the parser computes this prediction in two stages, first predicting the tag in the tag-word pair, and then predicting the word conditioned on that tag. We implement this conditioning as a mixture model (Bishop, 1995), where the tag predictions are the mixing coefficients. This means that only estimates for the tag-word pairs with the correct tag need to be computed, both in training and testing. We also reduced the computational cost of word prediction by replacing lower frequency tag-word pairs with a tag-"unknown-word" pair. Excluding words below a frequency threshold can greatly reduce the size of the vocabulary, because there are a very large number of low frequency words. This method also has the advantages of training an output to be used for words which were not in the training set, and smoothing across tag-word pairs whose low frequency would prevent accurate learning by themselves. We do not do any morphological analysis of unknown words, although we would expect some improvement in performance if we did. A variety of frequency thresholds were tried, as reported in section 6. The same representation of tag-word pairs was used in the input as was used for prediction.

6 The Experimental Results

The generality and efficiency of the above parsing model makes it possible to test a SSN parser on the Penn Treebank (Marcus et al., 1993), and thereby compare its performance directly to other statistical parsing models in the literature. To test the effects of varying vocabulary sizes on performance and tractability, we trained three different models. The simplest model ("Tags") includes no words in the vocabulary, relying completely on the information provided by the part-of-speech tags of the words. The second model ("Freq≥200") uses all tag-word pairs which occur at least 200 times in the training set. The remaining words were all treated as instances of the unknown-word. This resulted in a vocabulary size of 512 tag-word pairs. The third model ("Freq≥20") thresholds the vocabulary at 20 instances in the training set, resulting in 4242 tag-word pairs.[5]

As is standard practice, we used sections 2-22 as the training set (39,832 sentences), section 24 as a development/validation set (1346 sentence), and section 23 as a testing set (2416 sentences). We determined appropriate training parameters and network size based on our previous experience with networks similar to the models Tags and Freq≥200, which had been trained and evaluated on the same training and validation sets. We trained two or three networks for each of the three models and chose the best one based on their validation performance. We then tested the best non-lexicalized and the best lexicalized models on the testing set.[6] Standard measures of performance are shown in table 1.[7]

The top panel of table 1 lists the results for the non-lexicalized model (SSN–Tags) and the available results for three other models which only use part-of-speech tags as inputs, another neural network parser (Costa et al., 2001), an earlier statistical left-corner parser (Manning and Carpenter, 1997), and a PCFG (Charniak, 1997). The Tags model achieves performance which is better than any previously published results on parsing with a non-lexicalized model. The Tags model also does much better than the only other broad coverage neural network parser (Costa et al., 2001).

	Length≤40		All	
	LR	LP	LR	LP
Costa-et-al01	NA	NA	57.8	64.9
Manning&Carpenter97	77.6	79.9	NA	NA
Charniak97 (PCFG)	71.2	75.3	70.1	74.3
SSN–Tags	83.9	84.9	83.3	84.3
Ratnaparkhi99	NA	NA	86.3	87.5
Collins99	88.5	88.7	88.1	88.3
Charniak00	90.1	90.1	89.6	89.5
Collins00	90.1	90.4	89.6	89.9
Bod01	90.8	90.6	89.7	89.7
SSN–Freq≥200	88.8	89.6	88.3	89.2

Table 1: Percentage labeled constituent recall and precision on the testing set.

The bottom panel of table 1 lists the results for the chosen lexicalized model (SSN–Freq≥200) and five recent statistical parsers (Ratnaparkhi, 1999; Collins, 1999; Charniak, 2000; Collins, 2000; Bod, 2001). The performance of the lexicalized model falls in the middle of this range, only being beaten by the three best current parsers, which all achieve equivalent performance. The best current model (Collins, 2000) has only 6% less precision error and only 11% less recall error than the lexicalized model. The SSN parser achieves this result using much less lexical knowledge than other approaches, which all minimally use the words which occur at least 5 times, plus morphological features of the remaining words. It is also achieved without any explicit notion of lexical head.

7 Discussion and Further Analysis

Two novel aspects of this SSN parsing model are the small vocabulary size and the use of induced features to represent the derivation history. To investigate these aspects we trained some additional

[5] In these experiments the tags are included in the input to the system, but, for compatibility with other parsers, we did not use the hand-corrected tags which come with the corpus. We used a publicly available tagger (Ratnaparkhi, 1996) to tag the words and then used these in the input to the system.

[6] We found that 80 hidden units produced better performance than 60 or 100. Momentum was applied throughout training. Weight decay regularization was applied at the beginning of training but reduced to zero by the end of training.

[7] All our results are computed with the evalb program following the now-standard criteria in (Collins, 1999).

	Validation, Length≤100		
	LR	LP	$F_{\beta=1}$
Tags	82.9	84.2	83.6
Freq≥200	88.0	89.5	88.8
Freq≥20	87.9	89.2	88.5
Freq≥200, ancestor label	82.6	85.4	84.0
Freq≥200, child label	85.1	86.5	85.8
Freq≥200, lc–child label	86.1	87.8	86.9
Freq≥200, all labels	81.0	83.6	82.3

Table 2: Percentage labeled constituent recall, precision, and F-measure on the validation set.

models and tested them on the validation set.[8]

The first three rows of table 2 show performance as the vocabulary size increases from 46 tags (Tags) to 512 tag-word pairs (Freq≥200) to 4242 tag-word pairs (Freq≥20). There is a 32% reduction in F-measure error when we add words with frequency greater than 200, but the performance does not increase when we further increase the vocabulary size. Two explanations for the lack of improvement with larger vocabularies suggest themselves. The first possible explanation is that something about the parser design prevents the SSN from fully exploiting lexical information. One candidate for such a problem is the lack of any inductive bias expressing the importance of lexical heads over non-head words. The second possible explanation is that the importance of lexical items in previous models is mainly that they provide information about the types of contexts in which the lexical items tend to occur. This indirect representation of context is not very important if the model has a good representation of the actual context. The only lexical items which would then be necessary are the idiosyncratic ones, which tend to be high frequency. This argument is supported by the fact that the non-lexicalized model with induced history features actually does better than the lexicalized model without them (shown in the last line of table 2, and discussed below).

The last four rows of table 2 show the effects of reducing the use of induced history features. If when computing history representations, we remove a node from $D(top_i)$ and add its label to $f(d_1,..., d_{i-1})$, then we are replacing the induced features of the node's derivation history with the symbolic label of the node. This removes access to more distant characteristics of the node's derivation history. Table 2 shows the performance of models where this replacement is done for none (Freq≥200), one, or all of the nodes in $D(top_i)$ other than top_i itself. The biggest decrease in performance occurs when the left-corner ancestor's history representation is removed (ancestor label). This implies that more distant top-down constraints and constraints from the left context are playing a big role in the success of the SSN parser. Another big decrease in performance occurs when the most recent child's history representation is removed (child label). This implies that more distant bottom-up constraints are also playing a big role. There is also a decrease in performance when the left-corner child's history representation is remove (lc–child label). This implies that the first child does tend to carry information which is relevant throughout the subderivation for the node, and suggests that this child deserves a special status. Finally, not using any of these sources of induced history features (all labels) results in dramatically worse performance, with a 58% increase in F-measure error over using all three.

8 Conclusions

This paper has presented a statistical left-corner parser which uses a neural network to estimate the parameters of its generative probability model. A Simple Synchrony Network is trained to estimate the probabilities of parse decisions conditioned on the previous parse history, and these estimates are used to efficiently search for the most probable parse. When trained and tested on the standard Penn Treebank datasets, the parser's performance (88.8% F-measure) is within 1% of the best current parsers for this task, despite using a small vocabulary size (512 inputs).

This level of performance is achieved in large part due to Simple Synchrony Networks' ability to induce a finite representation of the unbounded parse history. By automatically inducing features of the parse history, this method avoids the need

[8] The validation set is used to avoid repeated testing on the standard testing set. The sentence with length greater than 100 was excluded. F-measure is $(2 \times LP \times LR)/(LP + LR)$.

to choose hand-crafted history features and their associated independence assumptions. Crucial to the success of this induction of history features is imposing biases which focus the induction process on structurally local aspects of the parse history. When the induced history features for structurally local aspects of the parse history are replaced by hand-crafted features (namely node labels), performance degrades dramatically. In addition to demonstrating the usefulness of a Simple Synchrony Network's induced history representation, this work also adds to the diversity of available broad coverage parsing methods (potentially of great interest for ensemble learning) and demonstrates the ability of neural network probability estimation to scale up to large datasets, unrestricted structures, and fairly large vocabularies.

References

Christopher M. Bishop. 1995. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, UK.

E. Black, F. Jelinek, J. Lafferty, D. Magerman, R. Mercer, and S. Roukos. 1993. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proc. 31st Meeting of Association for Computational Linguistics*, pages 31–37, Columbus, Ohio.

Rens Bod. 2001. What is the minimal set of fragments that achieves maximal parse accuracy? In *Proc. 34th Meeting of Association for Computational Linguistics*, pages 66–73.

Eugene Charniak. 1997. Statistical parsing with a context-free grammar and word statistics. In *Proc. 14th National Conference on Artificial Intelligence*, Providence, RI. AAAI Press/MIT Press.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proc. 1st Meeting of North American Chapter of Association for Computational Linguistics*, pages 132–139, Seattle, Washington.

Eugene Charniak. 2001. Immediate-head parsing for language models. In *Proc. 39th Meeting of Association for Computational Linguistics*, pages 116–223, Toulouse France.

Michael Collins and Nigel Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures and the voted perceptron. In *Proc. 35th Meeting of Association for Computational Linguistics*, pages 263–270.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia, PA.

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proc. 17th Int. Conf. on Machine Learning*, pages 175–182, Stanford, CA.

F. Costa, V. Lombardo, P. Frasconi, and G. Soda. 2001. Wide coverage incremental parsing by learning attachment preferences. In *Proc. of the Conf. of the Italian Association for Artificial Intelligence*.

James Henderson. 2000. A neural network parser that handles sparse data. In *Proc. 6th Int. Workshop on Parsing Technologies*, pages 123–134, Trento, Italy.

E.K.S. Ho and L.W. Chan. 1999. How to design a connectionist holistic parser. *Neural Computation*, 11(8):1995–2016.

K. Hornik, M. Stinchcombe, and H. White. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366.

Mark Johnson. 1998. PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632.

Peter Lane and James Henderson. 2001. Incremental syntactic parsing of natural language corpora with simple synchrony networks. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):219–231.

Christopher D. Manning and Bob Carpenter. 1997. Probabilistic parsing using left corner language models. In *Proc. Int. Workshop on Parsing Technologies*, pages 147–158.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *Proc. Conf. on Empirical Methods in Natural Language Processing*, pages 133–142, Univ. of Pennsylvania, PA.

Adwait Ratnaparkhi. 1999. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34:151–175.

Brian Roark and Mark Johnson. 1999. Efficient probabilistic top-down and left-corner parsing. In *Proc. 37th Meeting of Association for Computational Linguistics*, pages 421–428.

D.J. Rosenkrantz and P.M. Lewis. 1970. Deterministic left corner parsing. In *Proc. 11th Symposium on Switching and Automata Theory*, pages 139–152.