In Proc. 42nd Meeting of Association for Computational Linguistics (ACL 2004), Barcelona, Spain, 2004.

Discriminative Training of a Neural Network Statistical Parser

James HENDERSON
School of Informatics, University of Edinburgh
2 Buccleuch Place
Edinburgh EH8 9LW
United Kingdom
james.henderson@ed.ac.uk

Abstract

Discriminative methods have shown significant improvements over traditional generative methods in many machine learning applications, but there has been difficulty in extending them to natural language parsing. One problem is that much of the work on discriminative methods conflates changes to the learning method with changes to the parameterization of the problem. We show how a parser can be trained with a discriminative learning method while still parameterizing the problem according to a generative probability model. We present three methods for training a neural network to estimate the probabilities for a statistical parser, one generative, one discriminative, and one where the probability model is generative but the training criteria is discriminative. The latter model outperforms the previous two, achieving state-of-the-art levels of performance (90.1% F-measure on constituents).

1 Introduction

Much recent work has investigated the application of discriminative methods to NLP tasks, with mixed results. Klein and Manning (2002) argue that these results show a pattern where discriminative probability models are inferior to generative probability models, but that improvements can be achieved by keeping a generative probability model and training according to a discriminative optimization criteria. We show how this approach can be applied to broad coverage natural language parsing. Our estimation and training methods successfully balance the conflicting requirements that the training method be both computationally tractable for large datasets and a good approximation to the theoretically optimal method. The parser which uses this approach outperforms both a generative model and a discriminative model, achieving state-of-the-art levels of performance (90.1% F-measure on constituents).

To compare these different approaches, we use a neural network architecture called Simple Synchrony Networks (SSNs) (Lane and Henderson, 2001) to estimate the parameters of the probability models. SSNs have the advantage that they avoid the need to impose hand-crafted independence assumptions on the learning process. Training an SSN simultaneously trains a finite representations of the unbounded parse history and a mapping from this history representation to the parameter estimates. The history representations are automatically tuned to optimize the parameter estimates. This avoids the problem that any choice of hand-crafted independence assumptions may bias our results towards one approach or another. The independence assumptions would have to be different for the generative and discriminative probability models, and even for the parsers which use the generative probability model, the same set of independence assumptions may be more appropriate for maximizing one training criteria over another. By inducing the history representations specifically to fit the chosen model and training criteria, we avoid having to choose independence assumptions which might bias our results.

Each complete parsing system we propose consists of three components, a probability model for sequences of parser decisions, a Simple Synchrony Network which estimates the parameters of the probability model, and a procedure which searches for the most probable parse given these parameter estimates. This paper outlines each of these components, but more details can be found in (Henderson, 2003b), and, for the discriminative model, in (Henderson, 2003a). We also present the training methods, and experiments on the proposed parsing models.

2 Two History-Based Probability Models

As with many previous statistical parsers (Ratnaparkhi, 1999; Collins, 1999; Charniak, 2000), we use a history-based model of parsing. Designing a history-based model of parsing involves two steps, first choosing a mapping from the set of phrase structure trees to the set of parses, and then choosing a probability model in which the probability of each parser decision is conditioned on the history of previous decisions in the parse. We use the same mapping for both our probability models, but we use two different ways of conditioning the probabilities, one generative and one discriminative. As we will show in section 6, these two different ways of parameterizing the probability model have a big impact on the ease with which the parameters can be estimated.

To define the mapping from phrase structure trees to parses, we use a form of left-corner parsing strategy (Rosenkrantz and Lewis, 1970). In a left-corner parse, each node is introduced 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. Parsing a constituent starts by pushing the leftmost word w of the constituent onto the stack with a *shift(w)* action. Parsing a constituent ends by either introducing the constituent's parent nonterminal (labeled Y) with a *project(Y)* action, or attaching to the parent with an *attach* action.[1] A complete parse consists of a sequence of these actions, $d_1,..., d_m$, such that performing $d_1,..., d_m$ results in a complete phrase structure tree.

Because this mapping from phrase structure trees to sequences of decisions about parser actions is one-to-one, finding the most probable phrase structure tree is equivalent to finding the parse $d_1,..., d_m$ which maximizes $P(d_1,..., d_m|w_1,..., w_n)$. This probability is only nonzero if $\mathit{yield}(d_1,..., d_m) = w_1,..., w_n$, so we can restrict attention to only those parses which actually yield the given sentence. With this restriction, it is equivalent to maximize $P(d_1,..., d_m)$, as is done with our first probability model.

The first probability model is generative, because it specifies the joint probability of the input sentence and the output tree. This joint probability is simply $P(d_1,..., d_m)$, since the probability of the input sentence is included in the probabilities for the *shift(w_i)* decisions included in $d_1,..., d_m$. The probability model is then defined by using the chain rule for conditional probabilities to derive the probability of a parse as the multiplication of the probabilities of each decision d_i conditioned on that decision's prior parse history $d_1,..., d_{i-1}$.

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

The parameters of this probability model are the $P(d_i|d_1,..., d_{i-1})$. Generative models are the standard way to transform a parsing strategy into a probability model, but note that we are not assuming any bound on the amount of information from the parse history which might be relevant to each parameter.

The second probability model is discriminative, because it specifies the conditional probability of the output tree given the input sentence. More generally, discriminative models try to maximize this conditional probability, but often do not actually calculate the probability, as with Support Vector Machines (Vapnik, 1995). We take the approach of actually calculating an estimate of the conditional probability because it differs minimally from the generative probability model. In this form, the distinction between our two models is sometimes referred to as "joint versus conditional" (Johnson, 2001; Klein and Manning, 2002) rather than "generative versus discriminative" (Ng and Jordan, 2002). As with the generative model, we use the chain rule to decompose the entire conditional probability into a sequence of probabilities for individual parser decisions, where $\mathit{yield}(d_j,..., d_k)$ is the sequence of words w_i from the *shift(w_i)* actions in $d_j,..., d_k$.

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

Note that $d_1,..., d_{i-1}$ specifies $\mathit{yield}(d_1,..., d_{i-1})$, so it is sufficient to only add $\mathit{yield}(d_i,..., d_m)$ to the conditional in order for the entire input sentence to be included in the conditional. We will refer to the string $\mathit{yield}(d_i,..., d_m)$ as the lookahead string, because it represents all those words which have not yet been reached by the parse at the time when decision d_i is chosen. The parameters of this model differ from those of the generative model only in that they include the lookahead string in the conditional.

Although maximizing the joint probability is the same as maximizing the conditional probability, the fact that they have different parameters means that estimating one can be much harder than estimating the other. In general we would expect that estimating the joint probability would be harder than estimating the conditional probability, because the joint

[1] More details on the mapping to parses can be found in (Henderson, 2003b).

probability contains more information than the conditional probability. In particular, the probability distribution over sentences can be derived from the joint probability distribution, but not from the conditional one. However, the unbounded nature of the parsing problem means that the individual parameters of the discriminative model are much harder to estimate than those of the generative model.

The parameters of the discriminative model include an unbounded lookahead string in the conditional. Because these words have not yet been reached by the parse, we cannot assign them any structure, and thus the estimation process has no way of knowing what words in this string will end up being relevant to the next decision it needs to make. The estimation process has to guess about the future role of an unbounded number of words, which makes the estimate quite difficult. In contrast, the parameters of the generative model only include words which are either already incorporated into the structure, or are the immediate next word to be incorporated. Thus it is relatively easy to determine the significance of each word.

3 Estimating the Parameters with a Neural Network

The most challenging problem in estimating $P(d_i|d_1,...,d_{i-1},\mathit{yield}(d_i,...,d_m))$ and $P(d_i|d_1,...,d_{i-1})$ is that the conditionals include an unbounded amount of information. Both the parse history $d_1,...,d_{i-1}$ and the lookahead string $\mathit{yield}(d_i,...,d_m)$ grow with the length of the sentence. In order to apply standard probability estimation methods, we use neural networks to induce finite representations of both these sequences, which we will denote $h(d_1,...,d_{i-1})$ and $l(\mathit{yield}(d_i,...,d_m))$, respectively. The neural network training methods we use try to find representations which preserve all the information about the sequences which are relevant to estimating the desired probabilities.

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

$$P(d_i|d_1,...,d_{i-1},\mathit{yield}(d_i,...,d_m)) \approx \\ P(d_i|h(d_1,...,d_{i-1}),l(\mathit{yield}(d_i,...,d_m)))$$

Of the previous work on using neural networks for parsing natural language, by far the most empirically successful has been the work using Simple Synchrony Networks. Like other recurrent network architectures, SSNs compute a representation of an unbounded sequence by incrementally computing a representation of each prefix of the sequence. At each position i, representations from earlier in the sequence are combined with features of the new position i to produce a vector of real valued features which represent the prefix ending at i. This representation is called a hidden representation. It is analogous to the hidden state of a Hidden Markov Model. As long as the hidden representation for position $i-1$ is always used to compute the hidden representation for position i, any information about the entire sequence could be passed from hidden representation to hidden representation and be included in the hidden representation of that sequence. When these representations are then used to estimate probabilities, this property means that we are not making any a priori hard independence assumptions (although some independence may be learned from the data).

The difference between SSNs and most other recurrent neural network architectures is that SSNs are specifically designed for processing structures. When computing the history representation $h(d_1,...,d_{i-1})$, the SSN uses not only the previous history representation $h(d_1,...,d_{i-2})$, but also uses history representations for earlier positions which are particularly relevant to choosing the next parser decision d_i. This relevance is determined by first assigning each position to a node in the parse tree, namely the node which is on the top of the parser's stack when that decision is made. Then the relevant earlier positions are chosen based on the structural locality of the current decision's node to the earlier decisions' nodes. In this way, the number of representations which information needs to pass through in order to flow from history representation i to history representation j is determined by the structural distance between i's node and j's node, and not just the distance between i and j in the parse sequence. This provides the neural network with a linguistically appropriate inductive bias when it learns the history representations, as explained in more detail in (Henderson, 2003b).

When computing the lookahead representation $l(\mathit{yield}(d_i,...,d_m))$, there is no structural information available to tell us which positions are most relevant to choosing the decision d_i. Proximity in the string is our only indication of relevance. Therefore we compute $l(\mathit{yield}(d_i,...,d_m))$ by running a recurrent neural network backward over the string, so that the most recent input is the first word in the lookahead string, as discussed in more detail in (Henderson, 2003a).

Once it has computed $h(d_1,...,d_{i-1})$ and (for the discriminative model) $l(yield(d_i,...,d_m))$, the SSN uses standard methods (Bishop, 1995) to estimate a probability distribution over the set of possible next decisions d_i given these representations. This involves further decomposing the distribution over all possible next parser actions into a small hierarchy of conditional probabilities, and then using log-linear models to estimate each of these conditional probability distributions. The input features for these log-linear models are the real-valued vectors computed by $h(d_1,...,d_{i-1})$ and $l(yield(d_i,...,d_m))$, as explained in more detail in (Henderson, 2003b). Thus the full neural network consists of a recurrent hidden layer for $h(d_1,...,d_{i-1})$, (for the discriminative model) a recurrent hidden layer for $l(yield(d_i,...,d_m))$, and an output layer for the log-linear model. Training is applied to this full neural network, as described in the next section.

4 Three Optimization Criteria and their Training Methods

As with many other machine learning methods, training a Simple Synchrony Network involves first defining an appropriate learning criteria and then performing some form of gradient descent learning to search for the optimum values of the network's parameters according to this criteria. In all the parsing models investigated here, we use the on-line version of Backpropagation to perform the gradient descent. This learning simultaneously tries to optimize the parameters of the output computation and the parameters of the mappings $h(d_1,...,d_{i-1})$ and $l(yield(d_i,...,d_m))$. With multi-layered networks such as SSNs, this training is not guaranteed to converge to a global optimum, but in practice a network whose criteria value is close to the optimum can be found.

The three parsing models differ in the criteria the neural networks are trained to optimize. Two of the neural networks are trained using the standard maximum likelihood approach of optimizing the same probability which they are estimating, one generative and one discriminative. For the generative model, this means maximizing the total joint probability of the parses and the sentences in the training corpus. For the discriminative model, this means maximizing the conditional probability of the parses in the training corpus given the sentences in the training corpus. To make the computations easier, we actually minimize the negative log of these probabilities, which is called cross-entropy error. Minimizing this error ensures that training will converge to a neural network whose outputs are estimates of the desired probabilities.[2] For each parse in the training corpus, Backpropagation training involves first computing the probability which the current network assigns to that parse, then computing the first derivative of (the negative log of) this probability with respect to each of the network's parameters, and then updating the parameters proportionately to this derivative.[3]

The third neural network combines the advantages of the generative probability model with the advantages of the discriminative optimization criteria. The structure of the network and the set of outputs which it computes are exactly the same as the above network for the generative model. But the training procedure is designed to maximize the conditional probability of the parses in the training corpus given the sentences in the training corpus. The conditional probability for a sentence can be computed from the joint probability of the generative model by normalizing over the set of all parses $d'_1,...,d'_{m'}$ for the sentence.

$$P(d_1,...,d_m|w_1,...,w_n) = \frac{P(d_1,..,d_m)}{\sum_{d'_1,...,d'_{m'}} P(d'_1,..,d'_{m'})}$$

So, with this approach, we need to maximize this normalized probability, and not the probability computed by the network.

The difficulty with this approach is that there are exponentially many parses for the sentence, so it is not computationally feasible to compute them all. We address this problem by only computing a small set of the most probable parses. The remainder of the sum is estimated using a combination of the probabilities from the best parses and the probabilities from the partial parses which were pruned when searching for the best parses. The probabilities of pruned parses are estimated in such a way as to minimize their effect on the training process. For each decision which is part of some un-pruned parses, we calculate the average probability of generating the remainder of the sentence by these un-

[2] Cross-entropy error ensures that the minimum of the error function converges to the desired probabilities as the amount of training data increases (Bishop, 1995), so the minimum for any given dataset is considered an estimate of the true probabilities.

[3] A number of additional training techniques, such as regularization, are added to this basic procedure, as will be specified in section 6.

pruned parses, and use this as the estimate for generating the remainder of the sentence by the pruned parses. With this estimate we can calculate the sum of the probabilities for all the pruned parses which originate from that decision. This approach gives us a slight overestimate of the total sum, but because this total sum acts simply as a weighting factor, it has little effect on learning. What is important is that this estimate minimizes the effect of the pruned parses' probabilities on the part of the training process which occurs after the probabilities of the best parses have been calculated.

After estimating $P(d_1,...,d_m|w_1,...,w_n)$, training requires that we estimate the first derivative of (the negative log of) this probability with respect to each of the network's parameters. The contribution to this derivative of the numerator in the above equation is the same as in the generative case, just scaled by the denominator. The difference between the two learning methods is that we also need to account for the contribution to this derivative of the denominator. Here again we are faced with the problem that there are an exponential number of derivations in the denominator, so here again we approximate this calculation using the most probable parses.

To increase the conditional probability of the correct parse, we want to decrease the total joint probabilities of the incorrect parses. Probability mass is only lost from the sum over all parses because *shift*(w_i) actions are only allowed for the correct w_i. Thus we can decrease the total joint probability of the incorrect parses by making these parses be worse predictors of the words in the sentence.[4] The combination of training the correct parses to be good predictors of the words and training the incorrect parses to be bad predictors of the words results in prediction probabilities which are not accurate estimates, but which are good at discriminating correct parses from incorrect parses. It is this feature which gives discriminative training an advantage over generative training. The network does not need to learn an accurate model of the distribution of words. The network only needs to learn an accurate model of how words disambiguate previous parsing decisions.

When we apply discriminative training only to the most probable incorrect parses, we train the network to discriminate between the correct parse and those incorrect parses which are the most likely to be mistaken for the correct parse. In this sense our approximate training method results in optimizing the decision boundary between correct and incorrect parses, rather than optimizing the match to the conditional probability. Modifying the training method to systematically optimize the decision boundary (as in large margin methods such as Support Vector Machines) is an area of future research.

5 Searching for the most probable parse

The complete parsing system uses the probability estimates computed by the SSN to search for the most probable parse. The search incrementally constructs partial parses $d_1,...,d_i$ by taking a parse it has already constructed $d_1,...,d_{i-1}$ and using the SSN to estimate a probability distribution $P(d_i|d_1,...,d_{i-1},...)$ over possible next decisions d_i. These probabilities are then used to compute the probabilities for $d_1,...,d_i$. In general, the partial parse with the highest probability is chosen as the next one to be extended, but to perform the search efficiently it is necessary to prune the search space. The main pruning is that only a fixed number of the most probable derivations are allowed to continue past the shifting of each word. Setting this post-word beam width to 5 achieves fast parsing with reasonable performance in all models. For the parsers with generative probability models, maximum accuracy is achieved with a post-word beam width of 100.

6 The Experiments

We used the Penn Treebank (Marcus et al., 1993) to perform empirical experiments on the proposed parsing models. In each case the input to the network is a sequence of tag-word pairs.[5] We report results for three different vocabulary sizes, varying in the frequency with which tag-word pairs must occur in the training set in order to be included explicitly in the vocabulary. A frequency threshold of 200 resulted in a vocabulary of 508 tag-word pairs, a

[4] Non-prediction probability estimates for incorrect parses can make a small contribution to the derivative, but because pruning makes the calculation of this contribution inaccurate, we treat this contribution as zero when training. This means that non-prediction outputs are trained to maximize the same criteria as in the generative case.

[5] We used a publicly available tagger (Ratnaparkhi, 1996) to provide the tags. For each tag, there is an unknown-word vocabulary item which is used for all those words which are not sufficiently frequent with that tag to be included individually in the vocabulary (as well as other words if the unknown-word case itself does not have at least 5 instances). We did no morphological analysis of unknown words.

threshold of 20 resulted in 4215 tag-word pairs, and a threshold of 5 resulted in 11,993 tag-word pairs

For the generative model we trained networks for the 508 ("GSSN-Freq≥200") and 4215 ("GSSN-Freq≥20") word vocabularies. The need to calculate word predictions makes training times for the 11,993 word vocabulary very long, and as of this writing no such network training has been completed. The discriminative model does not need to calculate word predictions, so it was feasible to train networks for the 11,993 word vocabulary ("DSSN-Freq≥5"). Previous results (Henderson, 2003a) indicate that this vocabulary size performs better than the smaller ones, as would be expected.

For the networks trained with the discriminative optimization criteria and the generative probability model, we trained networks for the 508 ("DGSSN-Freq≥200") and 4215 ("DGSSN-Freq≥20") word vocabularies. For this training, we need to select a small set of the most probable incorrect parses. When we tried using only the network being trained to choose these top parses, training times were very long and the resulting networks did not outperform their generative counterparts. In the experiments reported here, we provided the training with a list of the top 20 parses found by a network of the same type which had been trained with the generative criteria. The network being trained was then used to choose its top 10 parses from this list, and training was performed on these 10 parses and the correct parse.[6] This reduced the time necessary to choose the top parses during training, and helped focus the early stages of training on learning relevant discriminations. Once the training of these networks was complete, we tested both their ability to parse on their own and their ability to re-rank the top 20 parses of their associated generative model ("DGSSN-. . ., rerank").

We determined appropriate training parameters and network size based on intermediate validation results and our previous experience.[7] We trained several networks for each of the GSSN models and chose the best ones based on their validation performance. We then trained one network for each of the DGSSN models and for the DSSN model. The best post-word beam width was determined on the validation set, which was 5 for the DSSN model and 100 for the other models.

To avoid repeated testing on the standard testing set, we first compare the different models with their performance on the validation set. Standard measures of accuracy are shown in table 1.[8] The largest accuracy difference is between the parser with the discriminative probability model (DSSN-Freq≥5) and those with the generative probability model, despite the larger vocabulary of the former. This demonstrates the difficulty of estimating the parameters of a discriminative probability model. There is also a clear effect of vocabulary size, but there is a slightly larger effect of training method. When tested in the same way as they were trained (for reranking), the parsers which were trained with a discriminative criteria achieve a 7% and 8% reduction in error rate over their respective parsers with the same generative probability model. When tested alone, these DGSSN parsers perform only slightly better than their respective GSSN parsers. Initial experiments on giving these networks exposure to parses outside the top 20 parses of the GSSN parsers at the very end of training did not result in any improvement on this task. This suggests that at least some of the advantage of the DSSN models is due to the fact that re-ranking is a simpler task than parsing from scratch. But additional experimental work would be necessary to make any definite conclusions about this issue.

For comparison to previous results, table 2 lists the results for our best model (DGSSN-Freq≥20, rerank)[9] and several other statistical parsers (Ratnaparkhi, 1999; Collins, 1999; Collins and Duffy, 2002; Charniak, 2000; Collins, 2000; Bod, 2003) on the entire testing set. Our best performing model is more accurate than all these previous models except (Bod, 2003). This DGSSN parser achieves this result using much less lexical knowledge than other approaches, which mostly use at least the

[6] The 20 candidate parses and the 10 training parses were found with post-word beam widths of 20 and 10, respectively, so these are only approximations to the top parses.

[7] All the best networks had 80 hidden units for the history representation (and 80 hidden units in the lookahead representation). Weight decay regularization was applied at the beginning of training but reduced to near 0 by the end of training. Training was stopped when maximum performance was reached on the validation set, using a post-word beam width of 5.

[8] All our results are computed with the evalb program following the standard criteria in (Collins, 1999), and using the standard training (sections 2–22, 39,832 sentences, 910,196 words), validation (section 24, 1346 sentence, 31507 words), and testing (section 23, 2416 sentences, 54268 words) sets (Collins, 1999).

[9] On sentences of length at most 40, the DGSSN-Freq≥20-rerank model gets 90.1% recall and 90.7% precision.

	LR	LP	$F_{\beta=1^*}$
DSSN-Freq$\geq$5	84.9	86.0	85.5
GSSN-Freq$\geq$200	87.6	88.9	88.2
DGSSN-Freq$\geq$200	87.8	88.8	88.3
GSSN-Freq$\geq$20	88.2	89.3	88.8
DGSSN-Freq$\geq$200, rerank	88.5	89.6	89.0
DGSSN-Freq$\geq$20	88.5	89.7	89.1
DGSSN-Freq$\geq$20, rerank	89.0	90.3	89.6

Table 1: Percentage labeled constituent recall (LR), precision (LP), and a combination of both ($F_{\beta=1}$) on validation set sentences of length at most 100.

	LR	LP	$F_{\beta=1^*}$
Ratnaparkhi99	86.3	87.5	86.9
Collins99	88.1	88.3	88.2
Collins&Duffy02	88.6	88.9	88.7
Charniak00	89.6	89.5	89.5
Collins00	89.6	89.9	89.7
DGSSN-Freq$\geq$20, rerank	89.8	90.4	90.1
Bod03	90.7	90.8	90.7

* $F_{\beta=1}$ for previous models may have rounding errors.

Table 2: Percentage labeled constituent recall (LR), precision (LP), and a combination of both ($F_{\beta=1}$) on the entire testing set.

words which occur at least 5 times, plus morphological features of the remaining words. However, the fact that the DGSSN uses a large-vocabulary tagger (Ratnaparkhi, 1996) as a preprocessing stage may compensate for its smaller vocabulary. Also, the main reason for using a smaller vocabulary is the computational complexity of computing probabilities for the *shift(w_i)* actions on-line, which other models do not require.

7 Related Work

Johnson (2001) investigated similar issues for parsing and tagging. His maximal conditional likelihood estimate for a PCFG takes the same approach as our generative model trained with a discriminative criteria. While he shows a non-significant increase in performance over the standard maximal joint likelihood estimate on a small dataset, because he did not have a computationally efficient way to train this model, he was not able to test it on the standard datasets. The other models he investigates conflate changes in the probability models with changes in the training criteria, and the discriminative probability models do worse.

In the context of part-of-speech tagging, Klein and Manning (2002) argue for the same distinctions made here between discriminative models and discriminative training criteria, and come to the same conclusions. However, their arguments are made in terms of independence assumptions. Our results show that these generalizations also apply to methods which do not rely on independence assumptions.

While both (Johnson, 2001) and (Klein and Manning, 2002) propose models which use the parameters of the generative model but train to optimize a discriminative criteria, neither proposes training algorithms which are computationally tractable enough to be used for broad coverage parsing. Our proposed training method succeeds in being both tractable and effective, demonstrating both a significant improvement over the equivalent generative model and state-of-the-art accuracy.

Collins (2000) and Collins and Duffy (2002) also succeed in finding algorithms for training discriminative models which balance tractability with effectiveness, showing improvements over a generative model. Both these methods are limited to reranking the output of another parser, while our trained parser can be used alone. Neither of these methods use the parameters of a generative probability model, which might explain our better performance (see table 2).

8 Conclusions

This article has investigated the application of discriminative methods to broad coverage natural language parsing. We distinguish between two different ways to apply discriminative methods, one where the probability model is changed to a discriminative one, and one where the probability model remains generative but the training method optimizes a discriminative criteria. We find that the discriminative probability model is much worse than the generative one, but that training to optimize the discriminative criteria results in improved performance. Performance of the latter model on the standard test set achieves 90.1% F-measure on constituents, which is the second best current accuracy level, and only 0.6% below the current best (Bod, 2003).

This paper has also proposed a neural network training method which optimizes a discriminative criteria even when the parameters being estimated are those of a generative probability model. This training method successfully satisfies the conflicting constraints that it be computationally tractable and that it be a good approximation to the theoretically optimal method. This approach contrasts with previous approaches to scaling up discriminative methods to broad coverage natural language parsing, which have parameterizations which depart substantially from the successful previous generative models of parsing.

References

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

Rens Bod. 2003. An efficient implementation of a new DOP model. In *Proc. 10th Conf. of European Chapter of the Association for Computational Linguistics*, Budapest, Hungary.

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.

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.

James Henderson. 2003a. Generative versus discriminative models for statistical left-corner parsing. In *Proc. 8th Int. Workshop on Parsing Technologies*, pages 115–126, Nancy, France.

James Henderson. 2003b. Inducing history representations for broad coverage statistical parsing. In *Proc. joint meeting of North American Chapter of the Association for Computational Linguistics and the Human Language Technology Conf.*, pages 103–110, Edmonton, Canada.

Mark Johnson. 2001. Joint and conditional estimation of tagging and parsing models. In *Proc. 39th Meeting of Association for Computational Linguistics*, pages 314–321, Toulouse, France.

Dan Klein and Christopher D. Manning. 2002. Conditional structure versus conditional estimation in NLP models. In *Proc. Conf. on Empirical Methods in Natural Language Processing*, pages 9–16, Univ. of Pennsylvania, PA.

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.

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.

A. Y. Ng and M. I. Jordan. 2002. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA. MIT Press.

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.

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

Vladimir N. Vapnik. 1995. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York.