In Proc. 8th Int. Workshop on Parsing Technologies (IWPT 2003).

GENERATIVE VERSUS DISCRIMINATIVE MODELS FOR STATISTICAL LEFT-CORNER PARSING

James Henderson
Department of Computer Science, University of Geneva, Genève, Switzerland
James.Henderson@cui.unige.ch

Abstract

We propose two statistical left-corner parsers and investigate their accuracy at varying speeds. The parser based on a generative probability model achieves state-of-the-art accuracy when sufficient time is available, but when high speed is required the parser based on a discriminative probability model performs better. Neural network probability estimation is used to handle conditioning on both the unbounded parse histories and the unbounded lookahead strings.

1 Introduction

Left-corner parsing combines the early use of grammatical constraints from top down parsing with the early use of lexical information from bottom up parsing in a way which results in efficient parsing of natural language sentences. For statistical parsing this efficiency is the result of being able to prune unpromising analyses early from the search for the most probable parse. We have found that at certain points in the parse it is possible to prune down to a beam of only 100 analyses for our best performing left-corner parsing model, whereas Collins' [Collins 1999] top down parsing model uses a search beam width of 10,000. Given that parsing speed is at least linear in the width of the search beam, this severe pruning has a big impact on the speed of a statistical parser.

In this paper we consider two different probability models which can be used for statistical left-corner parsing, one generative and one discriminative, and compare both their speed and their accuracy. The difference between these two models is in the way in which they handle words which the left-corner parse has not yet reached. The discriminative model uses lookahead to provide these words as part of the input to each parsing decision, thereby having early access to this information. The generative model predicts each word when it is reached by the parse, thereby delaying decisions based on a word until it has more information about the grammatical context in which the word appears. We argue that the generative model's parser should have higher accuracy, while the discriminative model's parser should be faster by allowing the use of a smaller search beam width. Experiments using a variety of search beam widths confirm these predictions, with the generative model achieving 88.1% recall and 89.5% precision but reaching this accuracy by pruning the search to a beam of 100 analyses after each word, and the discriminative model achieving only 83.8% recall and 85.1% precision but reaching this accuracy by pruning the search to a beam of just 5 analyses after each word. The experiments also show that the speed-accuracy curves of these two models cross, so that when high parsing

speed is required the discriminative model has higher accuracy than the generative model. The discriminative model also has the advantage that it is amenable to the application of margin-based classification methods, such as Support Vector Machines [Vapnik 1995], which future work could exploit to improve its accuracy.

The parameters of these two left-corner probability models are estimated using neural networks. Neural networks allow us to estimate probabilities conditioned on both unbounded parse histories and unbounded lookahead strings without making a priori independence assumptions. This helps avoid a priori assumptions which might bias our results towards one model or the other, thereby making our results more generally applicable.

After discussing the two probability models, we present how neural networks are used to estimate the parameters of each of these models. Then we present the design of the beam search used to search for the most probable parse, and discuss the experimental results with a variety of search beam widths.

2 The Two Probability Models

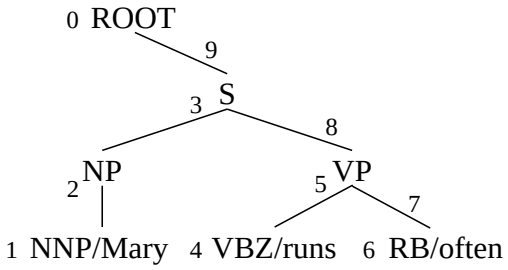
Both probability models are based on a form of left-corner parser [Rosenkrantz and Lewis 1970]. A left-corner parser decides to introduced a node into the parse tree after the subtree rooted at the node's first child has been fully parsed. Then the subtrees for the node's remaining children are parsed in their left-to-right order. We use a version of left-corner parsing which first applies right-binarization to the grammar, as is done in [Manning and Carpenter 1997] except that we binarize down to nullary rules rather than to binary rules. This allows the choice of the children for a node to be done incrementally, rather than all the children having to be chosen when the node is first introduced. This binarization makes this parsing strategy equivalent to PLR parsing [Soisalon-Soininen and Ukkonen 1979]. We also extended the parsing strategy slightly to handle Chomsky adjunction structures (i.e. structures of the form [X [X ...] [Y ...]]) as a special case. The Chomsky adjunction is removed and replaced with a special "modifier" link in the tree (becoming [X ... [$_{mod}$ Y ...]]). 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 ...]). All these grammar transforms are undone before any evaluation of the output trees is performed.

For the purposes of specifying our probability models, all we need to know about this parsing strategy is the ordering in which decisions about the parse tree are made. An example of this ordering is shown by the numbering on the left in figure 1. To precisely specify this ordering, it is sufficient to characterize the state of the parser as a stack of nodes which are in the process of being parsed, as illustrated on the right in figure 1. The parsing strategy starts with a stack that contains a node labeled $ROOT$ (step 0) and must end in the same configuration (step 9). Each parser action changes the stack and makes an associated specification of a characteristic of the parse tree. The possible parser actions are the following, where w is a tag-word pair, X, Y are nonterminal labels, and α is a stack of zero or more node labels.

shift(w) map stack α to α,w and specify that w is the next word in the sentence

project(Y) map stack α,X to α,Y and specify that Y is the parent of X in the tree

attach map stack α,Y,X to α,Y and specify that Y is the parent of X in the tree



Actions:	Stacks:
	0: ROOT
1: shift(NNP/Mary)	1: ROOT, NNP/Mary
2: project(NP)	2: ROOT, NP
3: project(S)	3: ROOT, S
4: shift(VBZ/runs)	4: ROOT, S, VBZ/runs
5: project(VP)	5: ROOT, S, VP
6: shift(RB/often)	6: ROOT, S, VP, RB/often
7: attach	7: ROOT, S, VP
8: attach	8: ROOT, S
9: attach	9: ROOT

Figure 1: The decomposition of a parse tree (left) into parse actions (center), and the parser's stack after each action (right).

modify map stack α,Y,X to α,Y and specify that Y is the modifier parent of X in the tree (i.e. X is Chomsky adjoined to Y)

A valid sequence of these parser actions is a derivation for a phrase structure tree, which we will denote $d_1, \ldots, d_m$. Because there is a one-to-one mapping from phrase structure trees to these derivations, we can define our two probability models in terms of these derivations.

$$P(\mathit{tree}(d_1, \ldots, d_m)|w_1, \ldots, w_n) = P(d_1, \ldots, d_m|w_1, \ldots, w_n)$$

Given a sentence $w_1, \ldots, w_n$, the objective of a statistical parser is to find the derivation $d_1, \ldots, d_m$ which maximizes $P(d_1, \ldots, d_m|w_1, \ldots, w_n)$. This probability is only nonzero if $\mathit{yield}(d_1, \ldots, d_m) = w_1, \ldots, w_n$, so we can restrict attention to only those derivations which actually yield the given sentence. With this restriction, it is equivalent to maximize $P(d_1, \ldots, d_m)$, 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 $P(d_1, \ldots, d_m)$, since the probability of the input sentence is included in the probabilities for the *shift(w_i)* actions. The probability model is then defined by using the chain rule for conditional probabilities to derive the probability of a derivation as the multiplication of the probabilities of each derivation decision d_i conditioned on that decision's prior derivation history $d_1, \ldots, d_{i-1}$.

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

The parameters of this probability model are the $P(d_i|d_1, \ldots, d_{i-1})$. 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 derivation history which might be relevant to each parameter. Thus there are an infinite number of parameters to the model, and this model cannot be converted into a probabilistic context free grammars.

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 is simpler and 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 actions.

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

Note that $d_1, \ldots, d_{i-1}$ specifies $yield(d_1, \ldots, d_{i-1})$, so it is sufficient to only add $yield(d_i, \ldots, d_m)$ to the conditional in order for the entire input sentence to be included in the conditional. We will refer to the string $yield(d_i, \ldots, d_m)$ as the lookahead string, because it represents all those words which have not yet been reached by the derivation at the time when action 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.

If we knew the exact values for all our parameters, then using the first probability model to maximize $P(d_1, \ldots, d_m)$ would give us exactly the same derivation as using the second probability model to maximize $P(d_1, \ldots, d_m | yield(d_1, \ldots, d_m))$. The difference between the two models arises because some parameters are easier to estimate than others, and thus some ways of parameterizing a model will result in better accuracy than other ways. The only difference between the parameters of the generative model and the parameters of the discriminative model is in the way in which they reflect the correlations between a parsing decision d_i and a word w which is shifted after d_i, at a step $j > i$. In the discriminative model, the first parameter to include both d_i and w is the probability for d_i, since w is in the lookahead string. Thus any nonzero correlations between d_i and w are reflected in the probability for d_i, which may be well before w appears in the derivation in d_j. In the generative model, the first parameter to include both d_i and w is the probability of the decision d_j to shift w. Thus any nonzero correlations between w and d_i will be reflected in the probability for d_j, which may be well after the decision d_i appears in the derivation. A simple example of this difference in figure 1 regards the obvious correlation between the NP projecting to an S in step 3 and the existence of a verb VBZ/runs in step 4. For the discriminative model this correlation would be reflected in the probability estimated for step 3 *P(project(S) | shift(NNP/Mary),project(NP), VBZ/runs,RB/often)*, while in the generative model it would be reflected in the probability estimated for step 4 *P(shift(VBZ/runs) | shift(NNP/Mary),project(NP),project(S))*.

The generative model makes it easier for the correlations between d_i and w to be estimated than the discriminative model, because at the point when they need to be estimated (step j), more is known about the structural relationship between the node where the decision d_i applies and the word w. For the discriminative model, all we know is that the word w is in a particular position in the lookahead string, so the estimate has to try to average over all possible structural relationships between the node where the decision d_i applies and w. We predict that the later estimate is much harder to learn accurately than the estimate where more is known, so we expect that the generative model will result in a more accurate parser than the discriminative model. This argument is closely related to that given in [Klein and Manning 2002] for why joint probability models perform better than conditional ones, except that their argument is made in terms of independence assumptions, whereas we are not assuming any a priori independence assumptions here.

The other prediction we can make about the difference between the two probability models

is in the speed at which they can parse. A major factor in determining the speed of a statistical parser is the extent to which the search for the most probable derivation can be pruned as the search proceeds. A statistical parser which uses the generative probability model cannot use any nonzero correlations between d_i and w to prune the search until w is shifted at step j. Between steps i and j the search must pursue multiple partial derivations, because it must choose between possible values of d_i before it reaches the decision d_j at which the constraints imposed by w on d_i become available. In contrast, a statistical parser which uses the discriminative probability model can use any nonzero correlations between d_i and w to prune the search at step i, thereby completely avoiding pursuing possible values of d_i which are incompatible with the constraints imposed by w on d_i.

From these arguments, we predict that there will be differences between the generative and discriminative models in both the accuracy and the speed of their statistical parsers. We predict that the generative model's statistical parser will have higher accuracy, because it is better able to learn correlations between a decision and any words which have not yet been shifted onto the stack. We predict that the discriminative model's statistical parser will have higher speed, because it will be better able to prune the search space down to a small number of partial derivations using these same correlations.

3 Estimating the Parameters of the Probability Models

We use neural networks to estimate the parameters $P(d_i|d_1,\ldots,d_{i-1})$ and $P(d_i|d_1,\ldots,d_{i-1},\mathit{yield}(d_i,\ldots,d_m))$.[1] In each case, the neural network performs this estimation in two stages, first computing a representation of the derivation history $d_1,\ldots,d_{i-1}$ (and the lookahead $\mathit{yield}(d_i,\ldots,d_m)$) and then computing a probability distribution over the possible decisions given that history (and lookahead). For the second stage we use standard neural network methods for probability estimation [Bishop 1995]. A log-linear model (also known as a maximum entropy model, and as the normalized exponential output function) is used to estimate the probability distribution over the four types of decisions, shifting, projecting, attaching, and modifying. A separate log-linear model is used to estimate the probability distribution over node labels given that projecting is chosen, which is multiplied by the probability estimate for projecting to get the probability estimates for that set of decisions.

For the discriminative model, once we have chosen to shift, the choice of what word to shift has probability 1 because it is already included in the conditional. For the generative model, the network must also compute a probability estimate for choosing to shift the word which is actually observed in the next position of the sentence. This is computed by multiplying the above estimate for shifting by the value for the observed word in an estimate of the probability distribution over all possible next words given that shifting is chosen. The latter estimate is also done with a log-linear model, which means that values for all possible words need to be computed. The high cost of this computation is reduced by also splitting this computation into multiple stages, first estimating a distribution over all possible tags, and then estimating a distribution over the possible tag-word pairs given the correct tag. This means that only estimates for the tag-word pairs with the correct tag need to be computed. We also reduced the cost of this computation by replacing the very large number of lower frequency tag-word

[1] The estimation method for the generative model is described in more detail in [Henderson 2003].

pairs with tag-"unknown-word" pairs, which are also used for tag-word pairs which were not in the training set. For the experiments reported here we used a high frequency threshold, as reported in section 5. Even so, the cost of computing the probability distribution over next words is high. The fact that the discriminative model does not require the computation of this distribution provides it with another speed advantage over the generative model, even before considering search beam widths.

The inputs to these log-linear models need to provide the necessary information about the derivation histories and lookahead strings in their probabilities' conditionals. The standard way to define these inputs would be to choose a priori a set of features for representing derivation histories and lookahead strings [Ratnaparkhi 1999, Charniak 2000]. This is difficult, because both the derivation histories and the lookahead strings are of unbounded length, but the set of features input to the log-linear models need to be of bounded size. The choice of features can have a large impact on accuracy because it implies an assumption of independence with the unbounded number of features which are not chosen. Also, any hand-crafted set of features could be criticized as being biased towards one probability model or the other, reducing the generality of our experimental results. The biases imposed by the a priori assumptions in previous studies comparing generative to discriminative (i.e. joint to conditional) probability models in NLP [Johnson 2001, Klein and Manning 2002] make it difficult to determine the generality of their results. To avoid such a priori assumptions, we use an automatic method for inducing finite representations of the unbounded derivation histories and lookahead strings. The induction method is the same for both models, but the actual representations which are induced are automatically tailored to the needs of the individual models.

The induction method is a form of multi-layered neural network called Simple Synchrony Networks [Lane and Henderson 2001, Henderson 2000]. The output layer of this network consists of the log-linear models discussed above. In addition, each SSN has a hidden layer which computes a finite vector of real valued features from a sequence of inputs specifying the derivation history $d_1, \ldots, d_{i-1}$. This hidden layer vector is an induced history representation $h(d_1, \ldots, d_{i-1})$. It is analogous to the hidden state of a Hidden Markov Model (HMM), in that it represents the state of the underlying generative process and in that it is not explicitly specified in the output of the generative process. Neural network training methods allow us to induce this history representation from a corpus of labeled training data.

The hidden layer computes $h(d_1, \ldots, d_{i-1})$ with a nonlinear function which allows the training to choose between a much larger set of mappings from inputs to outputs than is possible with a log-linear model alone.[2] In addition, the hidden layer computation is defined recursively, using one history representation $h(d_1, \ldots, d_{i-k-1})$ as part of the input to compute another history representation $h(d_1, \ldots, d_{i-1})$. This recursion allows the total input to the hidden layer computation to be unbounded, thereby allowing an unbounded derivation history to be successively compressed into a finite history representation. If the recursive compression of the derivation history is defined in such a way that at each step i the inputs to computing $h(d_1, \ldots, d_{i-1})$ include both the previous decision d_{i-1} and the previous history representation $h(d_1, \ldots, d_{i-2})$,

[2] Multi-layered neural networks can approximate arbitrary mappings from inputs to outputs [Hornik et al. 1989], whereas a log-linear model can only estimate probabilities where the category-conditioned probability distributions $P(x|d_i)$ of the pre-defined feature vectors x are in a restricted form of the exponential family [Bishop 1995].

then any information about the derivation history $d_1, \ldots, d_{i-1}$ could conceivably be included in $h(d_1, \ldots, d_{i-1})$. Thus such a model is making no a priori independence assumptions. However, information which has been input more recently in the recursion is much more likely to be included, so we can provide soft biases to the induction process by designing the flow of information through the recursive computation of history representations.

The principle we apply when designing the inputs to each history representation is that we want recency in this flow of information to match a linguistically appropriate notion of structural locality. We assign each derivation step i to the node top_i which is on the top of the stack before step i. The inputs to the history representation $h(d_1, \ldots, d_{i-1})$ at step i include the history representation for the most recent derivation step also assigned to top_i, plus the history representations for the most recent derivation steps assigned to 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, these inputs always include the history representation for the previous derivation step $i - 1$, as required to ensure that this model is making no a priori hard independence assumptions.

In addition to induced history features, the inputs at step i include hand-crafted features of the derivation history which are thought to be directly relevant to the decision to be made at step i. In the parser presented here, these inputs are the last decision d_{i-1} in the derivation, the label (or tag) of the step's node top_i, the tag-word pair for the most recently predicted word, and the tag-word pair for top_i's left-corner terminal (the leftmost word 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.

The discriminative model and the generative model both use the above method for inducing their history representations, but in addition the discriminative model induces a representation of the unbounded lookahead string. Here again we exploit the unbounded compression abilities of neural networks. We add to the discriminative model's SSN another hidden layer which computes a finite vector of real valued features $l(w_j, \ldots, w_n)$ from a sequence of inputs specifying the lookahead string. When this hidden layer computes the lookahead representation $l(w_j, \ldots, w_n)$ for position j it is given inputs specifying the tag-word pair w_j for position j and the lookahead representation $l(w_{j+1}, \ldots, w_n)$ for the next position $j + 1$. In other words, the tag-word pairs of the sentence are input to this hidden layer in reverse order, so that after the input of the tag-word pair for position j the hidden layer has a representation of the unbounded lookahead string from j to the end of the sentence. Because this results in information flowing backwards across the sentence, the induction of the lookahead representation is biased towards recording information about words which are earlier in the lookahead string. This is an appropriate bias because the earlier lookahead words are more likely to be relevant to the current parsing decision than the later ones.

The way these induced lookahead representations are used to compute outputs is designed

to put particular emphasis on the immediately following tag-word pair. When computing the outputs for a derivation step i which is after word w_{j-1} has been shifted and before word w_j has been shifted, we input the tag-word pair w_j directly to the hidden layer which computes the history representation $h(d_1, \ldots, d_{i-1})$, and we add the lookahead representation $l(w_{j+1}, \ldots, w_n)$ for word w_{j+1} to the set of features being used to compute the outputs at i.[3] The inductive bias provided by this arrangement of inputs means that the tag-word pair w_j is singled out for special treatment, particular attention is also payed to the tag-word pair w_{j+1}, and successively less attention is payed to tag-word pairs which are more distant in the future. Experiments indicate that this design is better than simply adding the lookahead representation $l(w_j, \ldots, w_n)$ to the set of features being used to compute the outputs.

4 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 derivation. Because we do not make a priori independence assumptions, searching the space of all possible derivations has exponential complexity, so it is important to be able to prune the search space if this computation is to be tractable. The left-corner ordering for derivations allows very severe pruning without significant loss in accuracy, which is crucial to the success of our parser due to the relatively high computational cost of computing probability estimates with a neural network rather than with the simpler methods typically employed in NLP. Our pruning strategy is designed specifically for left-corner parsing.

We prune the search space in two different ways, the first applying fixed beam pruning at certain derivation steps and the second restricting the branching factor at all derivation steps. The most important pruning occurs after each word has been shifted onto the stack. 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 shifting that word. Only a fixed beam of the best derivations are allowed to continue to the next word. We ran experiments with a variety of beam widths for this post-word pruning to explore the speed-accuracy tradeoff for the two probability models, as discussed in section 5. To search the space of derivations in between two words 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 5 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 5 was large enough that it had virtually no effect on the validation accuracy for either model, and in some cases it actually improved performance slightly over a branching factor of 10.

[3] This arrangement of inputs allows the network to compute the output using an arbitrary combination of the next tag-word pair w_j with the history representation $h(d_1, \ldots, d_{i-1})$, but assumes that the induced lookahead representation $l(w_{j+1}, \ldots, w_n)$ is such that the log-linear output computation is sufficient for combining with the history representations. Previous experience leads us to believe that this arrangement is sufficient.

5 The Experiments

We used the Penn Treebank [Marcus et al. 1993] to perform empirical experiments on these parsing models using the standard training (sections 2–22), validation (section 24), and testing (section 23) sets [Collins 1999]. We selected a single trained parser for each probability model and ran a series of experiments with different post-word beam widths, ranging from 1 to 150. The highest accuracy is state-of-the-art, being within 1% of the best current parser on this dataset [Collins 2000]; the generative model and a beam width of 100 achieved 88.3% labeled constituent recall, 89.2% precisions, and 88.8% F-measure on the testing set. Thus we believe that the overall results of these experiments are likely to generalize to other state-of-the-art methods of probability estimation.

The generative model's parser was selected after a number of experiments involving different vocabulary sizes, layer sizes, and training parameters. We found that using a vocabulary size of 512 tag-word pairs (thresholding at 200 instances in the training set) achieved an accuracy almost equivalent to using a vocabulary of 4242 tag-word pairs, but was much faster.[4] For this reason we chose to use the smaller vocabulary parser for these experiments.[5]

Having determined appropriate training parameters for the generative model, we used the same parameters to train three neural networks for the discriminative model. These networks differ from the generative one only in that they do not have output units for predicting the next word and they have additional hidden layers and inputs for computing the lookahead representation. The three networks differed in the size of the lookahead hidden layer, and we chose the best performing one based on validation accuracy.[6] Subsequent experiments with a larger vocabulary (12,088 tag-word pairs, thresholded at frequency 5) resulted in 1.1% higher recall and 0.9% higher precision, without any loss in parsing speed. However, this improvement had no impact on the overall pattern of results, so for the purposes of direct comparison we report here the results for the same sized vocabulary as the generative model.

We tested each of the selected parsers using a variety of post-word search beam widths and branching factors. To avoid repeated testing on the standard testing set, we used the validation set for these experiments. We removed the one sentence with length greater than 100 to make the results a better reflection of what they would be if we used the testing set. Standard measures of accuracy are plotted in figure 2 for different post-word beam widths.[7] In each case the plots show results for the lowest branching factor where maximum performance was achieved, which was 5 for both models.

The results shown in the right plot of figure 2 confirm the predictions made in section 2. The generative model's parser achieves higher accuracy, but requires a larger post-word beam width to achieve this accuracy. It does not reach its maximum accuracy until a post-word beam width of 100. But that accuracy is state-of-the-art at 88.1% recall and 89.5% precision. The discriminative model's parser reaches its maximum accuracy with a post-word beam width of

[4] These experiments are documented in [Henderson 2003]. For all experiments we used a publicly available tagger [Ratnaparkhi 1996] to provide the tags, rather than the hand-corrected tags which come with the corpus.

[5] The chosen network had 80 hidden units. training. Weight decay regularization was applied at the beginning of training but reduced to 0 by the end of training. Training was stopped when maximum accuracy was reached on the validation set using a post-word beam width of 5.

[6] The chosen network had 80 units in its lookahead hidden layer, although accuracy differed minimally from using 60 or 100 units. Validation accuracy was measured (for stopping training and for network selection) using a post-word beam width of 5.

[7] All our results are computed with the evalb program following the standard criteria in [Collins 1999].

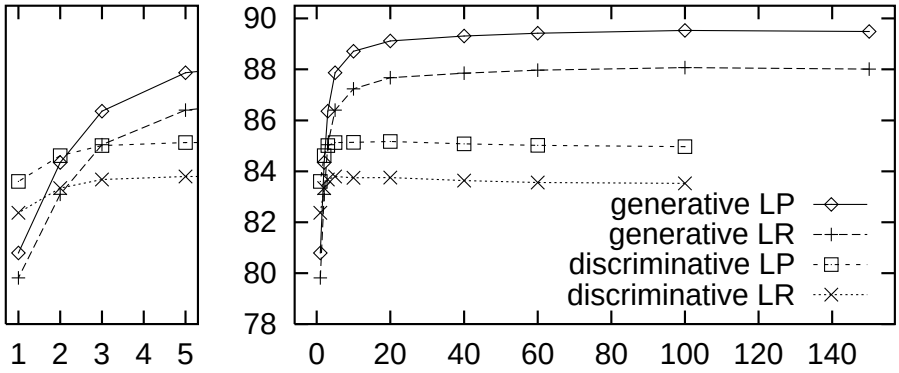


Figure 2: Labeled constituent recall (LR) and precision (LP) across different post-word beam widths for both models. The left plot zooms in on those points which are overlapping in the right plot.

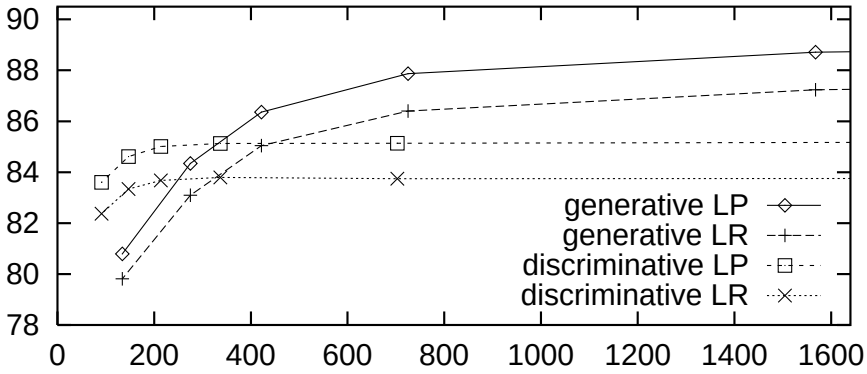


Figure 3: Labeled constituent recall (LR) and precision (LP) as a function of the number of seconds required to parse the dataset. The points vary in the post-word beam width used (1,2,3,5, and 10 for each curve).

only 5, but that accuracy is well below the state-of-the-art at 83.8% recall and 85.1% precision. When we compare accuracy between the two models at the same post-word beam width, the generative model performs better down to a beam width of 3. At the beam widths of 1 and 2 (shown in the left plot of figure 2) the discriminative model's parser has higher accuracy than the generative model's parser.

To get a better measure of the tradeoff between speed and accuracy for the two models, we also measured the speed at which the tests were computed by each model's parser. Elapsed time was measured for each test, run on a 502 MHz Sun Blade computer. Figure 3 shows labeled precision and recall plotted against the number of seconds required to parse all 31,392 words and 1345 sentences of the dataset. Figure 3 shows the points for the first five post-word beam widths shown in figure 2. As the plot shows, when parsing speeds of under 340 seconds (92 words per second) are required, better accuracy can be achieved by using the discriminative model. This break-even point also happens to be the maximum accuracy of the discriminative model, at a post-word beam width of 5. When the generative model reaches its maximum performance (with a post-word beam width of 100), it parses at a speed of 0.74 words per second, 125 times slower than the best discriminative model.

Except for the very low beam widths (where constant factors play a larger role), the speed of

the discriminative parser was over twice as fast as the generative parser for the same post-word beam width and branching factor. This is due to the fact that the discriminative model's parser does not have to compute word predictions. This difference would be much greater if a larger vocabulary size were used, resulting in a lower break-even speed for the speed-accuracy tradeoff. However, the main factor in determining speed is the beam width. We found that parser speed grew quadratically with post-word beam width, at $30x + 2x^2$ seconds for the discriminative model and $120x + 3x^2$ seconds for the generative model, both using a branching factor of 5.

6 Conclusions

In this paper we have presented two statistical left-corner parsers, one for a generative probability model and one for a discriminative probability model, and compared their accuracy and speed. We found that the generative model's parser achieved higher accuracy. We argue that this is due to it being better able to exploit grammatical constraints. We found that the discriminative model's parser requires a smaller search beam width to reach its maximum accuracy, which results in a much faster parser. We argue that this is due to the earlier use of lexical constraints. In addition we found that the two model's speed-accuracy curves cross, making the discriminative model a better choice when high speed is required, and the generative model a better choice when speed is less of an issue. Both these parser's benefit from the use of neural networks to do the statistical estimation, which allows the probabilities to be conditioned on both an unbounded derivation history and an unbounded lookahead string without making a priori independence assumptions.

References

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

[Charniak 2000] 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.

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

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

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

[Henderson 2003] James Henderson. 2003. 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.*, Edmonton, Canada.

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

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

[Johnson 2001] 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.

[Klein and Manning 2002] 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.

[Lane and Henderson 2001] 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.

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

[Marcus et al. 1993] 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.

[Ng and Jordan 2002] 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.

[Ratnaparkhi 1996] 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.

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

[Roark and Johnson 1999] 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.

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

[Soisalon-Soininen and Ukkonen 1979] E. Soisalon-Soininen and E. Ukkonen. 1979. A method for transforming grammars into LL(k) form. *Acta Informatica*, 12:339–369.

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