In Proc. North American Chap. Assoc. Computational Linguistics and Human Language Technology Conf. (HLT-NAACL 2003).

Inducing History Representations for Broad Coverage Statistical Parsing

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

Abstract

We present a neural network method for inducing representations of parse histories and using these history representations to estimate the probabilities needed by a statistical left-corner parser. The resulting statistical parser achieves performance (89.1% F-measure) on the Penn Treebank which is only 0.6% below the best current parser for this task, despite using a smaller vocabulary size and less prior linguistic knowledge. Crucial to this success is the use of structurally determined soft biases in inducing the representation of the parse history, and no use of hard independence assumptions.

1 Introduction

Unlike most problems addressed with machine learning, parsing natural language sentences requires choosing between an unbounded (or even infinite) number of possible phrase structure trees. The standard approach to this problem is to decompose this choice into an unbounded sequence of choices between a finite number of possible parser actions. This sequence is the parse for the phrase structure tree. We can then define a probabilistic model of phrase structure trees by defining a probabilistic model of each parser action in its parse context, and apply machine learning techniques to learn this model of parser actions.

Many statistical parsers (Ratnaparkhi, 1999; Collins, 1999; Charniak, 2000) are based on a history-based model of parser actions. In these models, the probability of each parser action is conditioned on the history of previous actions in the parse. But here again we are faced with an unusual situation for machine learning problems, conditioning on an unbounded amount of information. A major challenge in designing a history-based statistical parser is choosing a finite representation of the unbounded parse history from which the probability of the next parser action can be accurately estimated. Previous approaches have used a hand-crafted finite set of features to represent the parse history (Ratnaparkhi, 1999; Collins, 1999; Charniak, 2000). In the work presented here, we automatically induce a finite set of real valued features to represent the parse history.

We perform the induction of a history representation using an artificial neural network architecture, called Simple Synchrony Networks (SSNs) (Lane and Henderson, 2001; Henderson, 2000). This machine learning method is specifically designed for processing unbounded structures. It allows us to avoid making a priori independence assumptions, unlike with hand-crafted history features. But it also allows us to make use of our a priori knowledge by imposing structurally specified and linguistically appropriate biases on the search for a good history representation.

The combination of automatic feature induction and linguistically appropriate biases results in a history-based parser with state-of-the-art performance. When trained on just part-of-speech tags, the resulting parser achieves the best current performance of a non-lexicalized parser on the Penn Treebank. When a relatively small vocabulary of words is used, performance is only marginally below the best current parser accuracy. If either the biases are reduced or the induced history representations are replaced with hand-crafted features, performance degrades.

2 Estimating the Parameters of the Probability Model

The parsing system we propose consists of two components, one which estimates the parameters of a probability model for phrase structure trees, and one which searches for the most probable phrase structure tree given these parameters. The probability model we use is generative and history-based. At each step, the model's stochastic process generates a characteristic of the tree

or a word of 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, we can use the chain rule for conditional probabilities to derive the probability of a tree as the multiplication of the probabilities of each derivation decision d_i conditional on that decision's prior derivation history $d_1,...,d_{i-1}$.

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

The neural network is used to estimate the parameters $P(d_i|d_1,...,d_{i-1})$ of this 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 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. 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.

An example of the ordering of the decisions in a derivation 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 (steps 1, 4, and 6)

project(Y) map stack α,X to α,Y and specify that Y is the parent of X in the tree (steps 2, 3, and 5)

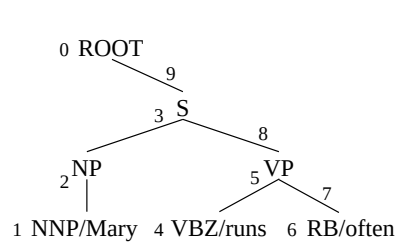

Figure 1: The decomposition of a parse tree into derivation decisions (left) and the stack after each decision (right). The derivation is *shift(NNP/Mary), project(NP), project(S), shift(VBZ/runs), project(VP), shift(RB/often), attach, attach, attach*.

attach map stack α,Y,X to α,Y and specify that Y is the parent of X in the tree (steps 7, 8, and 9)

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) (not illustrated)

Any valid sequence of these parser actions is a derivation $d_1,...,d_m$ for a phrase structure tree.

The neural network estimates the parameters $P(d_i|d_1,...,d_{i-1})$ in two stages, first computing a representation of the derivation history $h(d_1,...,d_{i-1})$ and then computing a probability distribution over the possible decisions given that history.

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

For the second stage, computing o_{d_i}, 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 $P(\textit{project}(X)|\textit{project},\alpha)$, which is multiplied by the probability estimate for projecting $P(\textit{project}|\alpha)$ to get the probability estimates for that set of decisions $P(\textit{project}(X)|\alpha)$.

$$P(\textit{project}(X)|\alpha) = P(\textit{project}(X)|\textit{project},\alpha)P(\textit{project}|\alpha)$$

Similarly, the probability estimate for shifting the word which is actually observed in the sentence $P(\textit{shift}(w)|\alpha)$ is computed with log-linear models. This means that values for all possible words need to be computed, to do the normalization. The high cost of this computation is reduced by splitting the computation of $P(\textit{shift}(w)|\textit{shift},\alpha)$ into multiple stages, first estimating a distribution over all possible tags $P(\textit{shift}(t/)|\textit{shift},\alpha)$, and then estimating a distribution over the possible tag-word pairs given the correct tag $P(\textit{shift}(t/w)|\textit{shift}(t/),\alpha)$.

$$P(\textit{shift}(t/w)|\alpha) = P(\textit{shift}(t/w)|\textit{shift}(t/), \alpha) P(\textit{shift}(t/)|\textit{shift}, \alpha) P(\textit{shift}|\alpha)$$

This means that only estimates for the tag-word pairs with the correct tag need to be computed. We also reduced the computational cost of terminal prediction by replacing the very large number of lower frequency tag-word pairs with tag-"unknown-word" pairs, which are also used for tag-word pairs which were not in the training set. 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.

3 Inducing a Representation of the Derivation History

The most novel aspect of our parsing model is the way in which the representation of the derivation history $h(d_1,...,d_{i-1})$ is computed. Choosing this representation is a challenge for any history-based statistical parser, because the history $d_1,...,d_{i-1}$ is of unbounded size. Log-linear models, as with 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 hand-craft a finite set of features which provides a sufficient summary of the 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.

In this work we use a method for automatically inducing a finite representation of the derivation history. The 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 is the log-linear model which computes the function o, discussed above. In addition the SSN has a hidden layer, which computes a finite vector of real valued features from a sequence of inputs specifying the derivation history $d_1,...,d_{i-1}$. This hidden layer vector is the history representation $h(d_1,...,d_{i-1})$. 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.

The mapping h from the derivation history to the history representation is computed with the recursive application of a function g. As will be discussed in the next section, g maps previous history representations $h(d_1,\ldots,d_{i-k-1})$ plus pre-defined features of the derivation history $f(d_1,...,d_{i-1})$ to a real-valued vector $h(d_1,\ldots,d_{i-1})$. Because the function g is nonlinear, the induction of this history representation 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. $o(x)$).[1] 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 h to be unbounded, because an unbounded derivation history can 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.[2] 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 representation. 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 representation. Given this difficulty, it is important to impose appropriate biases on the search for a good history representation.

4 The Inductive Bias on History Representations

When researchers choose a hand-crafted set of features to represent the derivation history, they are imposing a domain-dependent bias on the learning process through the independence assumptions which are implied by this choice. In this work we do not make any independence assumptions, but instead impose soft biases to emphasize some features of the derivation history over others. This is achieved through the choice of what features $f(d_1,...,d_{i-1})$ are input explicitly to the computation g of $h(d_1,...,d_{i-1})$ and what other history representations

[1] 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).

[2] 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.

$h(d_1,...,d_{i-k-1})$ are also input. If the explicit features include the previous decision d_{i-1} and the other history representations include the previous history representation $h(d_1,...,d_{i-2})$, then (by induction) any information about the derivation history $d_1,...,d_{i-1}$ could conceivably be included in $h(d_1,...,d_{i-1})$. Thus such a model is making no a priori independence assumptions. However, some of this information is more likely to be included than other of this information, which is the source of the model's soft biases.

The bias towards including certain information in the history representation arises from 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 selects a subset of the information from a set of previous history representations and records it in a new history representation. The training process automatically chooses the parameters of g 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. 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.

To achieve this structurally-determined inductive bias, we use Simple Synchrony Networks, which are specifically designed for processing structures. 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.

For each position i in the sub-derivation for a node top_i, the SSN computes two real-valued vectors, namely $h(d_1,...,d_{i-1})$ and $o(h(d_1,...,d_{i-1}))$. $h(d_1,...,d_{i-1})$ is computed by applying the function g to a set of predefined 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 particularly relevant to decisions involving top_i. This set 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 history representation to another, and thus determines the inductive bias of the system.

We have designed $D(top_i)$ and $f(d_1,...,d_{i-1})$ so that the inductive bias reflects structural locality. Thus, $D(top_i)$ 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 is 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 representations of both the left-corner ancestor and the most recent child, a derivation step i always has access to the history representation 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 representation. 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 representation 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 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 the best 100 partial derivations which end in shifting that word. Only a fixed beam of the best 100 derivations are allowed to continue to the next word. Experiments with a variety of post-prediction beam widths confirms that very small validation performance gains are achieved with widths larger than 100. 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 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 the validation accuracy.

6 The Experimental Results

We used the Penn Treebank (Marcus et al., 1993) to perform empirical experiments on this parsing model. To test the effects of varying vocabulary sizes on performance and tractability, we trained three different models. The simplest model ("SSN-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 ("SSN-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 ("SSN-Freq≥20") thresholds the vocabulary at 20 instances in the training set, resulting in 4242 tag-word pairs. [3]

	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
SSN-Freq≥20	89.3	89.9	88.8	89.5

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

We determined appropriate training parameters and network size based on intermediate validation results and our previous experience with networks similar to the models SSN-Tags and SSN-Freq≥200. We trained two or three networks for each of the three vocabulary sizes and chose the best ones based on their validation performance. Training times vary but are long, being around 4 days for a SSN-Tags model, 6 days for a SSN-Freq≥200 model, and 10 days for a SSN-Freq≥20 model (on a 502 MHz Sun Blade computer). We then tested the best models for each vocabulary size on the testing set.[4] Standard measures of performance are shown in table 1.[5]

[3] We used a publicly available tagger (Ratnaparkhi, 1996) to provide the tags used in these experiments, rather than the hand-corrected tags which come with the corpus.

[4] All the best networks had 80 hidden units. Weight decay regularization was applied at the beginning of training but reduced to 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.

[5] 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), validation (section 24, 1346 sentence), and testing (section 23, 2416 sentences) sets (Collins, 1999).

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 SSN-Tags model achieves performance which is much better than the only other broad coverage neural network parser (Costa et al., 2001). The SSN-Tags model also does better than any other published results on parsing with just part-of-speech tags, as exemplified by the results for (Manning and Carpenter, 1997) and (Charniak, 1997).

The bottom panel of table 1 lists the results for the two lexicalized models (SSN-Freq≥200 and SSN-Freq≥20) and five recent statistical parsers (Ratnaparkhi, 1999; Collins, 1999; Charniak, 2000; Collins, 2000; Bod, 2001). On the complete testing set, the performance of our lexicalized models is very close to the three best current parsers, which all achieve equivalent performance. The performance of the best current parser (Collins, 2000) represents only a 4% reduction in precision error and only a 7% reduction in recall error over the SSN-Freq≥20 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.

Another diffference between the three best parsers and ours is that we parse incrementally using a beam search. This allows use to trade off parsing accuracy for parsing speed, which is a much more important issue than training time. Running times to achieve the above levels of performance on the testing set averaged around 30 seconds per sentence for SSN-Tags, 1 minute per sentence for SSN-Freq≥200, and 2 minutes per sentence for SSN-Freq≥20 (on a 502 MHz Sun Blade computer, average 22.5 words per sentence). But by reducing the number of alternatives considered in the search for the most probable parse, we can greatly increase parsing speed without much loss in accuracy. With the SSN-Freq≥200 model, accuracy slightly better than (Collins, 1999) can be achieved at 2.7 seconds per sentence, and accuracy slightly better than (Ratnaparkhi, 1999) can be achieved at 0.5 seconds per sentence (Henderson, 2003) (on validation sentences at most 100 words long, average 23.3 words per sentence).

7 Discussion and Further Analysis

To investigate the role which induced history representations are playing in this parser, we trained a number of additional SSNs and tested them on the validation set.[6] The middle panel of table 2 shows the performance when some of the induced history representations are replaced with the label of their associated node. The first four lines show the performance when this replacement is performed individually for each of the history representations input to the computation of a new history representation, namely that for the node's left-corner ancestor, its most recent child, its left-corner child, and the previous parser action at the node itself, respectively. The final line shows the performance when all these replacements are done. In the first two models this replacement has the effect of imposing a hard independence assumption in place of the soft biases towards ignoring structurally more distant information. This is because there is no other series of history representations through which the removed information could pass. In the second two models this replacement simply removes the bias towards paying attention to more structurally local information, without imposing any independence assumptions.

	Validation, Length≤100		
	LR	LP	$F_{\beta=1}$
SSN-Freq≥200	88.0	89.5	88.8
ancestor label	82.6	85.4	84.0
child label	85.1	86.5	85.8
lc-child label	86.1	87.8	86.9
self label	86.8	88.1	87.4
all labels	78.8	82.2	80.5
head identification	87.1	88.6	87.9
head word	87.6	89.0	88.3
head word and child	87.1	88.7	87.9

Table 2: Percentage labeled constituent recall, precision, and F-measure on the validation set for different versions of the SSN-Freq≥200 model.

In each modified model there is a reduction in performance, as compared to the case where all these history representations are used (SSN-Freq≥200). The biggest decrease in performance occurs when the left-corner ancestor is represented with just its label (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, and suggests that parsers which do not include information about this context in their history features will not do well. Another big decrease in performance occurs when the most recent child is represented with just its label (child label). This implies that more distant bottom-up constraints are also playing a big role, probably including some information

[6] The validation set is used to avoid repeated testing on the standard testing set. Sentences of length greater than 100 were excluded.

about lexical heads. There is also a decrease in performance when the left-corner child is represented with just its label (lc-child label). This implies that the first child does tend to carry information which is relevant throughout the sub-derivation for the node, and suggests that this child deserves a special status in a history representation. Interestingly, a smaller, although still substantial, degradation occurs when the previous history representation for the same node is replaced with its node label. We suspect that this is because the same information can be passed via its children's history representations. 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.

One bias which is conspicuously absent from our parser design is a bias towards paying particular attention to lexical heads. The concept of lexical head is central to theories of syntax, and has often been used in designing hand-crafted history features (Collins, 1999; Charniak, 2000). Thus it is reasonable to suppose that the incorporation of this bias would improve performance. On the other hand, the SSN may have no trouble in discovering the concept of lexical head itself, in which case incorporating this bias would have little effect.

To investigate this issue, we trained several SSN parsers with an explicit representation of phrasal head. Results are shown in the lower panel of table 2. The first model (head identification) includes a fifth type of parser action, *head_attach*, which is used to identify the head child of each node in the tree. Although incorrectly identifying the head child does not effect the performance for these evaluation measures, forcing the parser to learn this identification results in some loss in performance, as compared to the SSN-Freq≥200 model. This is to be expected, since we have made the task harder without changing the inductive bias to exploit the notion of head. The second model (head word) uses the identification of the head child to determine the lexical head of the phrase.[7] After the head child is attached to a node, the node's lexical head is identified and that word is added to the set of features $f(d_1,...,d_{i-1})$ input directly to the node's subsequent history representations. This adds an inductive bias towards treating the lexical head as important for post-head parsing decisions. The results show that this inductive bias does improve performance, but not enough to compensate for the degradation caused by having to learn to identify head children. The lack of a large improvement suggests that the SSN-Freq≥200 model already learns the significance of lexical heads, but perhaps a different method for incorporating the bias towards conditioning on lexical heads could improve performance a little. The third model (head word and child) extends the head word model by adding the head child to the set of structurally local nodes $D(top_i)$. This addition does not result in an improvement, suggesting that the induced history representations can identify the significance of the head child without the need for additional bias. The degradation appears to be caused by increased problems with overtraining, due to the large number of additional weights.

[7] If a node's head child is a word, then that word is the node's lexical head. If a node's head child is a nonterminal, then the lexical head of the head child is the node's lexical head.

8 Related Work

Most previous work on statistical parsing has used a history-based probability model with a hand-crafted set of features to represent the derivation history (Ratnaparkhi, 1999; Collins, 1999; Charniak, 2000). Ratnaparkhi (1999) defines a very general set of features for the histories of a shift-reduce parsing model, but the results are not as good as models which use a more linguistically informed set of features for a top-down parsing model (Collins, 1999; Charniak, 2000). In addition to the method proposed in this paper, another alternative to choosing a finite set of features is to use kernel methods, which can handle unbounded feature sets. However, this causes efficiency problems. 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). Future work could use the induced history representations from our work to define efficiently computable tree kernels.

The only other broad coverage neural network parser (Costa et al., 2001) also uses a neural network architecture which is specifically designed for processing structures. We believe that their poor performance is due to a network design which does not take into consideration the recency bias discussed in section 4. Ratnaparkhi's parser (1999) can also be considered a form of neural network, but with only a single layer, since it uses a log-linear model to estimate its probabilities. Previous work on applying SSNs to natural language parsing (Henderson, 2000) has not been general enough to be applied to the Penn Treebank, so it is not possible to compare results directly to this work.

9 Conclusions

This paper has presented a method for estimating the parameters of a history-based statistical parser which does not require any a priori independence assumptions. A neural network is trained simultaneously to estimate the probabilities of parser actions and to induce a finite repre-

sentation of the unbounded parse history. The probabilities of parser actions are conditioned on this induced history representation, rather than being conditioned on a set of hand-crafted history features chosen a priori. A beam search is used to search for the most probable parse given the neural network's probability estimates. When trained and tested on the standard Penn Treebank datasets, the parser's performance (89.1% F-measure) is only 0.6% below the best current parsers for this task, despite using a smaller vocabulary and less prior linguistic knowledge.

The neural network architecture we use, Simple Synchrony Networks, not only allows us to avoid imposing hard independence assumptions, it also allows us to impose linguistically appropriate soft biases on the learning process. SSNs are specifically designed for processing structures, which allows us to design the SSN so that the induced representations of the parse history are biased towards recording structurally local information about the parse. When we modify these biases so that some structurally local information tends to be ignored, performance degrades. When we introduce independence assumptions by cutting off access to information from more distant parts of the structure, performance degrades dramatically. On the other hand, we find that biasing the learning to pay more attention to lexical heads does not improve performance.

References

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

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.

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.

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

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.