In Proc. 13th Int. Conf. on Artificial Neural Networks (ICANN/ICONIP 2003).

Structural Bias in Inducing Representations for Probabilistic Natural Language Parsing

James Henderson

Dept. of Computer Science, University of Geneva, Genève, Switzerland
James.Henderson@cui.unige.ch,
WWW home page: http://cui.unige.ch/~henderson/

Abstract. We present a neural network based natural language parser. Training the neural network induces hidden representations of unbounded partial parse histories, which are used to estimate probabilities for parser decisions. This induction process is given domain-specific biases by matching the flow of information in the network to structural locality in the parse tree, without imposing any independence assumptions. The parser achieves performance on the benchmark datasets which is roughly equivalent to the best current parsers.

1 Introduction

Processing structured data, and particularly natural language parsing, has been a challenge for artificial neural networks. Recurrent neural network architectures are potentially useful in this domain, due to their ability to recursively compress unbounded structures into a finite hidden representation. But this potential has been muted by their strong bias towards only including recent information in a hidden representation and ignoring information earlier in the recursion. This bias has prevented many attempts at parsing with neural networks from scaling up to long sentences with large parse trees [1], and is probably responsible for the poor performance when neural networks have been applied to broad coverage parsing [2]. Rather than trying to avoid this bias, in this work we exploit this bias to help the network's training induce hidden representations which are appropriate to the domain. We propose a method for neural network structure processing which matches recency in the network's recursive computation to a domain-dependent notion of locality in the structure.

We apply this locality principle to the design of Simple Synchrony Networks (SSNs) [3, 4] for estimating the probabilities of parser decisions. The resulting statistical parsers achieve performance roughly equivalent to the state-of-the-art. Performance with part-of-speech tags as input is better than any other such parser. With words as input, performance (89.1% F-measure) is only 0.6% below the best current parser, despite using a relatively small vocabulary.

2 Inducing History Representations with SSNs

Natural language parsing takes a string of words of unbounded length (the sentence) and produces a tree structure of unbounded size (the parse). In statistical

parsing, the objective is to estimate the probability of each possible tree structure for the input sentence, and choose the most probable one. To handle the unbounded size of the trees, the probability of a parse tree is broken down into a sequence of probabilities for individual decisions about the tree, such as choosing the label of a node in the tree or deciding which node is the parent of another node. This sequence of decisions is called the tree's derivation. Assuming that there is a one-to-one mapping between trees and 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 $d_1,...,d_{i-1}$.

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

Given such a probability model, we want to design a neural network for estimating the model's parameters $P(d_i|d_1,...,d_{i-1})$.

The most challenging problem in estimating $P(d_i|d_1,...,d_{i-1})$ is how to represent the unbounded amount of information in the derivation history $d_1,...,d_{i-1}$. We would like to have a finite representation $h(d_1,...,d_{i-1})$ of this history. The standard practise in statistical parsing is to make a priori independence assumptions which allow us to ignore all the information about $d_1,...,d_{i-1}$ except a finite set of history features [5, 6]. Holistic approaches to neural network parsing avoid such independence assumptions by applying a neural network architecture for sequence processing directly to the history sequence $d_1,...,d_{i-1}$ [1]. Training automatically induces a hidden representation which is used for $h(d_1,...,d_{i-1})$. We take a similar approach, but use a neural network architecture, Simple Synchrony Networks [3, 4], which is capable of exploiting both the sequential ordering of the derivation history and the underlying structural nature of the tree which the derivation specifies.

SSNs allow us to exploit the underlying tree structure because they do not treat a derivation as a single holistic sequence, but as a set of sub-derivations. Each one of these sub-derivations is associated with a node in the tree, and structural relationships between nodes are used to determine how information is passed from one of these sub-derivations to another. The unbounded number of nodes in a parse tree is not a problem for this approach, because SSNs can process an unbounded set of sub-derivations, as well as handling the unbounded length of each sub-derivation.

A SSN processes each of the sub-derivations in its set in the same way as a Simple Recurrent Network (SRN) [7]. At each step $s_{j,k}$ in the sub-derivation s_j for node j, a context layer is used to record the hidden layer activations from the previous step $s_{j,k-1}$ in s_j, and these activations plus the new inputs at step $s_{j,k}$ are used to compute new hidden layer activations and the outputs for step $s_{j,k}$. The vector of new hidden layer activations at step $s_{j,k}$ is the history representation $h(d_1,...,d_{s_{j,k}-1})$ for the derivation decision $d_{s_{j,k}}$. The outputs for step $s_{j,k}$ are the estimates for the probability distribution $P(d_{s_{j,k}}|d_1,...,d_{s_{j,k}-1})$. The new inputs at position $i = s_{j,k}$ are a set of pre-defined features of the derivation history $f(d_1,...,d_{i-1})$ and a set of history representations $\{rep_{i-1}(l)|l \in D(j)\}$ for nodes $D(j)$ which are in pre-defined structural relationships to node j,

where $rep_{i-1}(l) = h(d_1,...,d_{\max(k|k\leq i \wedge N(k)=l)})$ is the most recent previous history representation assigned to node l, and $N(k)$ is the node for the sequence to which step k is assigned.

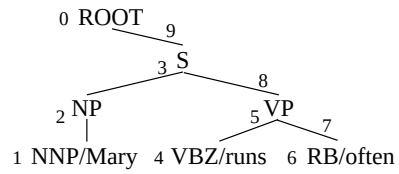
To avoid making independence assumptions we need $f(d_i|d_1,...,d_{i-1})$ to always include d_{i-1} and $D(N(i))$ to always include N(i-1). In this case, any information about the derivation history $d_1,...,d_{i-1}$ could in principle be included in the history representation $h(d_1,...,d_{i-1})$ (by induction). However, in practice some information is much more likely to be included than other information. During training, the error which is backpropagated through the network's recursive computation quickly vanishes as it passes from hidden layer computation to hidden layer computation. This means that training will tend to ignore correlations between inputs and outputs which are separated by many hidden layer computations, and focus on correlations between inputs and outputs which are close together. However, this inductive bias can actually be very useful. Inducing a finite hidden representation of an unboundedly large history is a very difficult optimization problem, so it is important to bias the learning towards representations which we know a priori to be good ones.

We achieve this bias by placing inputs which we know to be relevant to a given output close to that output in the flow of information between history representations. Features of the derivation history $d_1,...,d_{i-1}$ which are expected to be directly relevant to the derivation decision d_i are included in the predefined inputs $f(d_1,...,d_{i-1})$. And if two derivation decisions d_i and d_{i+k} are expected to be dependent on the same features of the derivation history, then the node $N(i)$ for the earlier decision is included in the set of nodes $D(N(i_k))$ whose history representations are input to the computation for the later decision. This approach differs from the approach of making independence assumptions primarily in that these biases are soft while independence assumptions are hard. Sufficiently strong correlations which contradict our expectations will still be discovered by the training.

3 The SSN Statistical Parser

The complete parsing system uses the probability estimates computed by the SSN to search for the derivation $d_1,...,d_m$ with the highest probability $P(d_1,...,d_m)$. The search incrementally constructs partial derivations $d_1,...,d_i$ by taking a derivation it has already constructed $d_1,...,d_{i-1}$, using the SSN to estimate a probability distribution $P(d_i|d_1,...,d_{i-1})$ over possible next decisions d_i, and computing $P(d_1,...,d_i) = P(d_1,...,d_{i-1})P(d_i|d_1,...,d_{i-1})$. In general, the partial derivation 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 (see [8] for details). We find that the search is still tractable with sufficiently little pruning that it has virtually no effect on parser accuracy.

The parameters $P(d_i|d_1,...,d_{i-1})$ which the SSN estimates are determined by the order in which derivation decisions are made. The derivation ordering which we use here is that of a form of left-corner parser [9], illustrated by the numbering



Stacks:

0: ROOT	5: ROOT, S, VP
1: ROOT, NNP/Mary	6: ROOT, S, VP, RB/often
2: ROOT, NP	7: ROOT, S, VP
3: ROOT, S	8: ROOT, S
4: ROOT, S, VBZ/runs	9: ROOT

Fig. 1. The order in which aspects of the parse tree are specified in a derivation (left) and the derivation's stack after each decision (right).

in the left half of figure 1. The parsing of a subtree proceeds bottom-up from its leftmost terminal, and introduces the node for the root of the subtree after the root's first child has been parsed. This root node is then placed on a stack, where it provides top-down information for the parsing of the root's remaining children, as illustrated in the right half of figure 1. More detail about the derivations used can be found in [8].

The SSN uses standard methods [10] to estimate a probability distribution over the set of possible next decisions d_i given the SSN's history representation $h(d_1,..., d_{i-1})$. Due to the need to choose node labels and predict input words, there are a very large number of possible next decisions. For this reason we separate the computation of $P(d_i|d_1,..., d_{i-1})$ into several sub-computations, and use mixture models to combine them. Each sub-computation uses the normalized exponential output function. The first computation has one output for each of the three basic types of decisions: predict the next tag-word pair $P(predict|\alpha)$, introduce a new node $P(introduce|\alpha)$, or pop the stack $P(pop|\alpha)$, where $\alpha = h(d_1,..., d_{i-1})$. The second computation has one output for each node label.

$$P(introduce(l)|\alpha) = P(introduce(l)|introduce, \alpha)P(introduce|\alpha)$$

When computing the probability of predicting the next word, we avoid the need to normalize over the entire set of words by first computing a probability distribution over tags, and then computing a probability distribution over tag-word pairs conditioned on the tag.

$$P(predict(t/w)|\alpha) = P(predict(t/w)|predict(t/), \alpha)P(predict(t/)|predict, \alpha)P(predict|\alpha)$$

This means that only the tag-word computation for the correct tag needs to be performed. We also reduced the computational cost of word prediction by replacing the very large number of lower frequency tag-word pairs with a tag-"unknown-word" pair. This method also has the advantages of training an output to be used for words which were not in the training set, and smoothing across tag-word pairs whose low frequency would prevent accurate learning by themselves. A variety of frequency thresholds were tried, as reported in section 5. The inputs for tag-word pairs also include separate inputs for tags and the same tag-"unknown-word" pairs.

4 The Linguistically Appropriate Inductive Bias

As discussed in section 2, we can impose domain-dependent biases on the induction of history representations through the choice of $N(i)$, $D(N(i))$, and

$f(d_1,...,d_{i-1})$. In particular, we want information from the history representations for the nodes $D(N(i))$ to be relevant to the history representation for the node $N(i)$. We match recency in this flow of information between history representations to a linguistically appropriate notion of structural locality.

We assign a derivation step i to the node $N(i) = top_i$ which is on the top of the stack prior to that step. This means that $h(d_1,...,d_{i-1})$ will always receive as input the history representation of the most recent previous step which also had top_i on the top of the stack. This 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 future decisions involving top_i as well. As a simple example, in figure 1, the prediction of the leftmost 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).

Given that $N(i) = top_i$, we want to define $D(N(i))$ to be a set of nodes which are structurally local to top_i. We define $D(N(i))$ to be the left-corner ancestor of top_i (which is below top_i on the stack), top_i's leftmost child, and top_i's most recent child (which is top_{i-1}, or none). The history representations for these nodes always include that for the previous derivation step $i-1$, so we are not imposing any independence assumptions by this choice of $D(N(i))$.

The pre-defined features $f(d_1,...,d_{i-1})$ which are input directly to $h(d_1,...,d_{i-1})$ are those which we think are 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 sub-derivation's node top_i, the most recently predicted tag-word pair, and the tag-word pair for top_i's leftmost terminal.

5 The Experimental Results

We tested this SSN parsing model on the standard corpus (the Penn Treebank [11]), thereby allowing us to compare its performance directly to other broad coverage parsing models in the literature. To test the effects of varying the vocabulary size on performance and tractability, we trained three different models. The simplest model (SSN-Tags) includes no words in the vocabulary, using only part-of-speech tags as input. The second model (SSN-200) uses all tag-word pairs which occur at least 200 times in the training set, plus the tag-"unknown-word" pairs. This resulted in a vocabulary size of 512 tag-word pairs. The third model (SSN-20) thresholds the vocabulary at 20 instances in the training set, resulting in 4242 tag-word pairs.[1]

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-200. We trained two or three networks for each of the three models and chose the best one based on their validation performance.

[1] We used a publicly available tagger [12] to provide the tags used in these experiments.

	Cost01	Char97	SSN-Tags	Ratn99	Coll99	Coll00	SSN-200	SSN-20
recall	57.8	70.1	83.3	86.3	88.1	89.6	88.3	88.8
precision	64.9	74.3	84.3	87.5	88.3	89.9	89.2	89.5
F-measure	61.0	72.1	83.8	86.9	88.2	89.7	88.8	89.1

Table 1. Standard performance measures on labeled constituents in the testing set.

We then tested the best models for each vocabulary size on the testing set.[2] Standard measures of performance are shown in table 1.[3]

The left panel of table 1 lists the results for the SSN-Tags model and for two other models which only use part-of-speech tags as inputs: another neural network parser (Cost01) [2], and a probabilistic context free grammar (Char97) [13]. The SSN-Tags model achieves performance which is much better than the only other broad coverage neural network parser, Cost01. 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 Char97.

The right panel of table 1 lists the results for the two SSN models which use words, SSN-200 and SSN-20, and three recent statistical parsers: Ratn99 [5], Coll99 [6], and Coll00 [14]. The performance of the SSN models fall near the top of this range, only being beaten by the best current parsers, which all achieve performance roughly equivalent to Coll00. The best current parser, Coll00, has only 4% less precision error, only 7% less recall error, and only 6% less F-measure error than the best SSN model. The SSN parser achieves this performance using a much smaller vocabulary than other approaches, which all use a frequency threshold of at most 5 plus morphological analysis, and without any explicit notion of a phrase's head word, which has proved to be important in other models [6].

6 Related Work

Much previous work on parsing with neural networks has taken the approach of modeling the derivation sequence directly [1]. The problem with this approach is that when a parse tree is flattened into a single derivation sequence, some decisions which are structurally local in the tree will inevitably end up being far apart in the derivation. This implies an inductive bias in the neural network which is not appropriate for the domain, as discussed in section 2. As the parse tree grows larger the distance in the derivation between adjacent decisions in the tree also grows larger, which we believe is the main reason this approach has not been successfully scaled up beyond small parse trees [1].

[2] 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.

[3] All our results are computed following the standard criteria in [6], and using the standard training, validation, and testing sets [6]. F-measure is a combination of recall and precision. The F-measures values for previous models may have rounding errors.

Two neural network architectures have been developed specifically for application to structured representations, SSNs and Recursive Neural Networks (RNNs) [15]. The RNN architecture models structures by mapping the nodes and edges of the structure's graph directly to hidden layers and links in the neural network. When applied to natural language parsing [2], this mapping is applied to the partial parse trees produced during an incremental parse of a sentence. The RNN learns to estimate the probability that a partial parse tree is correct. For the purposes of broad coverage parsing the results are not very good, as shown in table 1 under Cost01. Based on the arguments made in section 2, we believe this is because the distance between an input and the output in the RNN's recursive computation is a function of the structural distance from the input's node to the tree's root, rather than the structural distance between the input's node and the node where a decision needs to be made.

The Simple Synchrony Network architecture was originally motivated by a neuroscientific hypothesis about how the timing of neuron activation spikes is used to represent objects [16]. The units of an SSN pulse, which provides them with two temporal dimensions for encoding information, period and phase. Phases are used to represent objects, and periods are used to represent the time course of computation. The key property of this architecture from the point of view of learning is that timing is used to represent objects, so the same link weights can be used for each object, resulting in the ability to generalize to unbounded sets of objects [3]. In this paper the objects are the nodes of the tree, so each phase includes the hidden and output unit activations for one node's sub-derivation. There is one period for each word in the sentence, but unlike in previous work, there may be periods in which a phase has no output and simply copies its previous hidden activations. We use the simplest version of the SSN architecture, which has no holistic representation of state (non-pulsing units) [3], only the hidden activations for individual sub-derivations. In this form the SSN architecture is a special case of the RNN architecture, but the RNN would be applied to the graph of the information flow between SSN sub-derivations, rather than to the tree structures as is done in [2]. Previous work on applying SSNs to parsing [4] has not been general enough to be applied to the standard corpus, so it is not possible to compare results directly to this work.

7 Conclusions

This paper has presented the design of a Simple Synchrony Network for statistical parsing. The SSN is trained to estimate the probabilities of derivation decisions conditioned on the previous derivation history, and these estimates are used to search for the most probable parse. When trained and tested on the standard datasets, the parser's performance (89.1% F-measure) is only 0.6% less than the best current parser [14], despite using a smaller vocabulary size (4242 inputs) and less prior linguistic knowledge.

This level of performance is achieved due to the Simple Synchrony Networks' success in inducing a good hidden representation of the unbounded derivation

history. Crucial to this success is the use of domain-dependent biases which focus the induction process on structurally local aspects of the derivation history. In addition to demonstrating the usefulness of this approach to neural network structure processing, this work demonstrates the ability of neural network structure processing to successfully scale up to unrestricted structures, large datasets, and fairly large vocabularies.

References

1. E.K.S. Ho and L.W. Chan. How to design a connectionist holistic parser. *Neural Computation*, 11(8):1995–2016, 1999.
2. F. Costa, V. Lombardo, P. Frasconi, and G. Soda. Wide coverage incremental parsing by learning attachment preferences. In *Proc. of the Conf. of the Italian Association for Artificial Intelligence*, 2001.
3. Peter Lane and James Henderson. Incremental syntactic parsing of natural language corpora with simple synchrony networks. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):219–231, 2001.
4. James Henderson. A neural network parser that handles sparse data. In *Proc. 6th Int. Workshop on Parsing Technologies*, pages 123–134, Trento, Italy, 2000.
5. Adwait Ratnaparkhi. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34:151–175, 1999.
6. Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing.* PhD thesis, University of Pennsylvania, Philadelphia, PA, 1999.
7. Jeffrey L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7:195–225, 1991.
8. James Henderson. 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, 2003.
9. D.J. Rosenkrantz and P.M. Lewis. Deterministic left corner parsing. In *Proc. 11th Symposium on Switching and Automata Theory*, pages 139–152, 1970.
10. Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Oxford, UK, 1995.
11. Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.
12. Adwait Ratnaparkhi. 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, 1996.
13. Eugene Charniak. Statistical parsing with a context-free grammar and word statistics. In *Proc. 14th National Conference on Artificial Intelligence*, Providence, RI, 1997. AAAI Press/MIT Press.
14. Michael Collins. Discriminative reranking for natural language parsing. In *Proc. 17th Int. Conf. on Machine Learning*, pages 175–182, Stanford, CA, 2000.
15. P. Frasconi, M. Gori, and A. Sperduti. A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks*, 9:768–786, 1998.
16. Lokendra Shastri and Venkat Ajjanagadde. From simple associations to systematic reasoning: A connectionist representation of rules, variables, and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences*, 16:417–451, 1993.