A Connectionist Architecture for Learning to Parse

James Henderson and **Peter Lane**
Dept of Computer Science, Univ of Exeter
Exeter EX4 4PT, United Kingdom
jamie@dcs.ex.ac.uk, pclane@dcs.ex.ac.uk

Abstract

We present a connectionist architecture and demonstrate that it can learn syntactic parsing from a corpus of parsed text. The architecture can represent syntactic constituents, and can learn generalizations over syntactic constituents, thereby addressing the sparse data problems of previous connectionist architectures. We apply these Simple Synchrony Networks to mapping sequences of word tags to parse trees. After training on parsed samples of the Brown Corpus, the networks achieve precision and recall on constituents that approaches that of statistical methods for this task.

1 Introduction

Connectionist networks are popular for many of the same reasons as statistical techniques. They are robust and have effective learning algorithms. They also have the advantage of learning their own internal representations, so they are less constrained by the way the system designer formulates the problem. These properties and their prevalence in cognitive modeling has generated significant interest in the application of connectionist networks to natural language processing. However the results have been disappointing, being limited to artificial domains and oversimplified subproblems (e.g. (Elman, 1991)). Many have argued that these kinds of connectionist networks are simply not computationally adequate for learning the complexities of real natural language (e.g. (Fodor and Pylyshyn, 1988), (Henderson, 1996)).

Work on extending connectionist architectures for application to complex domains such as natural language syntax has developed a theoretically motivated technique called Temporal Synchrony Variable Binding (Shastri and Ajjanagadde, 1993; Henderson, 1996). TSVB allows syntactic constituency to be represented, but to date there has been no empirical demonstration of how a learning algorithm can be effectively applied to such a network. In this paper we propose an architecture for TSVB networks and empirically demonstrate its ability to learn syntactic parsing, producing results approaching current statistical techniques.

In the next section of this paper we present the proposed connectionist architecture, Simple Synchrony Networks (SSNs). SSNs are a natural extension of Simple Recurrent Networks (SRNs) (Elman, 1991), which are in turn a natural extension of Multi-Layered Perceptrons (MLPs) (Rumelhart et al., 1986). SRNs are an improvement over MLPs because they generalize what they have learned over words in different sentence positions. SSNs are an improvement over SRNs because the use of TSVB gives them the additional ability to generalize over constituents in different structural positions. The combination of these generalization abilities is what makes SSNs adequate for syntactic parsing.

Section 3 presents experiments demonstrating SSNs' ability to learn syntactic parsing. The task is to map a sentence's sequence of part of speech tags to either an unlabeled or labeled parse tree, as given in a preparsed sample of the Brown Corpus. A network input-output format is developed for this task, along with some linguistic assumptions that were used to simplify these initial experiments. Although only a small training set was used, an SSN achieved 63% precision and 69% recall on unlabeled constituents for previously unseen sentences. This is approaching the 75% precision and recall achieved on a similar task by Probabilistic Context Free Parsers (Charniak, forthcoming), which is the best current method for parsing based on part of speech tags alone. Given that these are the very first results produced with this method, future developments are likely to improve on them, making the future for this method very promising.

2 A Connectionist Architecture that Generalizes over Constituents

Simple Synchrony Networks (SSNs) are designed to extend the learning abilities of standard connectionist networks so that they can learn generalizations over linguistic constituents. This generalization ability is provided by using Temporal Synchrony Variable Binding (TSVB) (Shastri and Ajjanagadde, 1993) to represent constituents. With TSVB, gener-
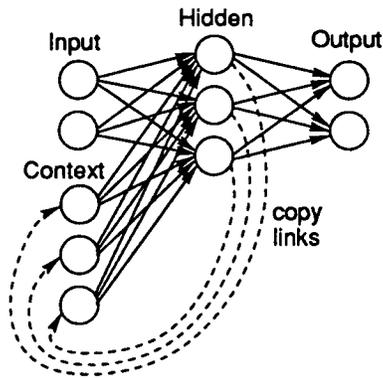
Figure 1: A Simple Recurrent Network.

alization over constituents is achieved in an exactly analogous way to the way Simple Recurrent Networks (SRNs) (Elman, 1991) achieve generalization over the positions of words in a sentence. SRNs are a standard connectionist method for processing sequences. As the name implies, SSNs are one way of extending SRNs with TSVB.

2.1 Simple Recurrent Networks

Simple Recurrent Networks (Elman, 1991) are a simple extension of the most popular form of connectionist network, Multi-Layered Perceptrons (MLPs) (Rumelhart et al., 1986). MLPs are popular because they can approximate any finite mapping, and because training them with the Backpropagation learning algorithm (Rumelhart et al., 1986) has been demonstrated to be effective in a wide variety of applications. Like MLPs, SRNs consist of a finite set of units which are connected by weighted links, as illustrated in figure 1. The output of a unit is simply a scalar activation value. Information is input to a network by placing activation values on the input units, and information is read out of a network by reading off activation values from the output units. Computation is performed by the input activation being scaled by the weighted links and passed through the activation functions of the "hidden" units, which are neither part of the input nor output. The only parameters in this computation are the weights of the links and how many hidden units are used. The number of hidden units is chosen by the system designer, but the link weights are automatically trained using a set of example input-output mappings and the Backpropagation learning algorithm.

Unlike MLPs, SRNs process sequences of inputs and produce sequences of outputs. To store information about previous inputs, SRNs use a set of context units, which simply record the activations of the hidden units during the previous time step (shown as dashed links in figure 1). When the SRN is done computing the output for one input in the sequence, the vector of activations on the hidden units is copied to the context units. Then the next input is processed with this copied pattern in the context units. Thus the hidden pattern computed for one input is used to represent the context for the subsequent input. Because the hidden pattern is learned, this method allows SRNs to learn their own internal representation of this context. This context is the state of the network. A number of algorithms exist for training such networks with loops in their flow of activation (called recurrence), for example Backpropagation Through Time (Rumelhart et al., 1986).

The most important characteristic of any learning-based model is the way it generalizes from the examples it is trained on to novel testing examples. In this regard there is a crucial difference between SRNs and MLPs, namely that SRNs generalize across sequence positions. At each position in a sequence a new context is copied, a new input is read, and a new output is computed. However the link weights that perform this computation are the same for all the positions in the sequence. Therefore the information that was learned for an input and context in one sequence position will inherently be generalized to inputs and contexts in other sequence positions. This generalization ability is manifested in the fact that SRNs can process arbitrarily long sequences; even the inputs at the end, which are in sequence positions that the network has never encountered before, can be processed appropriately. This generalization ability is a direct result of SRNs using time to represent sequence position.

Generalizing across sequence positions is crucial for syntactic parsing, since a word tends to have the same syntactic role regardless of its absolute position in a sentence, and there is no practical bound on the length of sentences. However this ability still doesn't make SRNs adequate for syntactic parsing. Because SRNs have a bounded number of output units, and therefore an effectively bounded output for each input, the space of possible outputs should be linear in the length of the input. For syntactic parsing, the total number of constituents is generally considered to be linear in the length of the input, but each constituent has to choose its parent from amongst all the other constituents. This gives us a space of possible parent-child relationships that is proportional to the square of the length of the input. For example, the attachment of a prepositional phrase needs to be chosen from all the constituents on the right frontier of the current parse tree. There may be an arbitrary number of these constituents, but an SRN would have to distinguish between them using only a bounded number of output units. While in theory such a representation can be achieved using arbitrary precision continuous activation values, this

bounded nature is symptomatic of a limitation in SRNs' generalization abilities. What we really want is for the network to learn what kinds of constituents such prepositional phrases like to attach to, and apply these generalizations independently of the absolute position of the constituent in the parse tree. In other words, we want the network to generalize over constituents. There is no apparent way for SRNs to achieve such generalization. This inability to generalize results in the network having to be trained on a set of sentences in which every kind of constituent appears in every position in the parse tree, resulting in serious sparse data problems. We believe that it is this difficulty that has prevented the successful application of SRNs to syntactic parsing.

2.2 Simple Synchrony Networks

The basic technique which we use to solve SRNs' inability to generalize over constituents is exactly analogous to the technique SRNs use to generalize over sentence positions; we process constituents one at a time. Words are still input to the network one at a time, but now within each input step the network cycles through the set of constituents. This dual use of time does not introduce any new complications for learning algorithms, so, as for SRNs, we can use Backpropagation Through Time. The use of timing to represent constituents (or more generally entities) is the core idea of Temporal Synchrony Variable Binding (Shastri and Ajjanagadde, 1993). Simple Synchrony Networks are an application of this idea to SRNs.[1]

As illustrated in figure 2, SSNs use the same method of representing state as do SRNs, namely context units. The difference is that SSNs have two of these memories, while SRNs have one. One memory is exactly the same as for SRNs (the figure's lower recurrent component). This memory has no representation of constituency, so we call it the "gestalt" memory. The other memory has had TSVB applied to it (the figure's upper recurrent component, depicted with "stacked" units). This memory only represents information about constituents, so we call it the constituent memory. These two representations are then combined via another set of hidden units to compute the network's output. Because the output is about constituents, these combination and output units have also had TSVB applied to them.

The application of TSVB to the output units allows SSNs to solve the problems that SRNs have with representing the output of a syntactic parser. For every step in the input sequence, TSVB units cycle through the set of constituents. To output

---

[1] There are a variety of ways to extend SRNs using TSVB. The architecture presented here was selected based on previous experiments using a toy grammar.
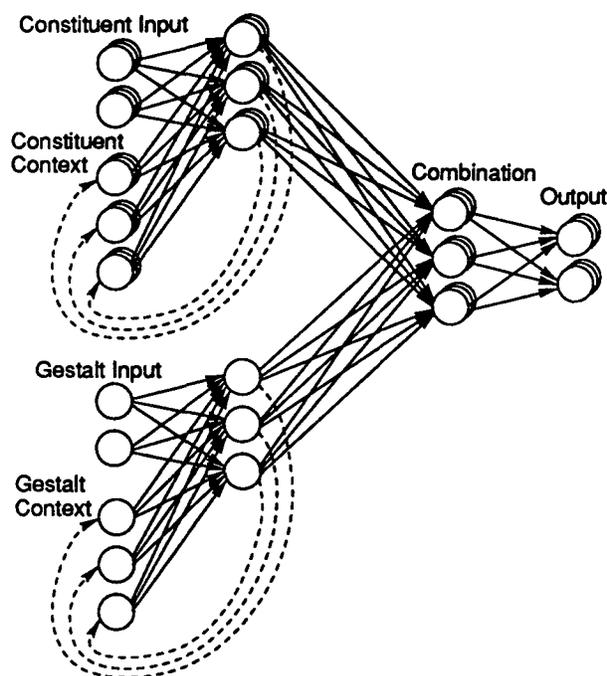

Figure 2: A Simple Synchrony Network. The units to which TSVB has been applied are depicted as several units stacked on top of each other, because they store activations for several constituents.

something about a particular constituent, it is simply necessary to activate an output unit at that constituent's time in the cycle. For example, when a prepositional phrase is being processed, the constituent which that prepositional phrase attaches to can be specified by activating a "parent" output unit in synchrony with the chosen constituent. However many constituents there are for the prepositional phrase to choose between, there will be that many times in the cycle that the "parent" unit can be activated in. Thereby we can output information about an arbitrary number of constituents using only a bounded number of units. We simply require an arbitrary amount of time to go through all the constituents.

Just as SRNs' ability to input arbitrarily long sentences was symptomatic of their ability to generalize over sentence position, the ability of SSNs to output information about arbitrarily many constituents is symptomatic of their ability to generalize over constituents. Having more constituents than the network has seen before is not a problem because outputs for the extra constituents are produced on the same units by the same link weights as for other constituents. The training that occurred for the other constituents modified the link weights so as to produce the constituents' outputs appropriately, and now these same link weights are applied to the

533

extra constituents. So the SSN has generalized what it has learned over constituents. For example, once the network has learned what kinds of constituents a preposition likes to attach to, it can apply these generalizations to each of the constituents in the current parse and choose the best match.

In addition to their ability to generalize over constituents, SSNs inherit from SRNs the ability to generalize over sentence positions. By generalizing in both these ways, the amount of data that is necessary to learn linguistic generalizations is greatly reduced, thus addressing the sparse data problems which we believe are the reasons connectionist networks have not been successfully applied to syntactic parsing. The next section empirically demonstrates that SSNs can be successfully applied to learning the syntactic parsing of real natural language.

3 Experiments in Learning to Parse

Adding the theoretical ability to generalize over linguistic constituents is an important step in connectionist natural language processing, but theoretical arguments are not sufficient to address the empirical question of whether these mechanisms are effective in learning to parse real natural language. In this section we present experiments on training Simple Synchrony Networks to parse naturally occurring sentences. First we present the input-output format for the SSNs used in these experiments, then we present the corpus, then we present the results, and finally we discuss likely future developments. Despite the fact that these are the first such experiments to be designed and run, an SSN achieved 63% precision and 69% recall on constituents. Because these results are approaching the results for current statistical methods for parsing from part of speech tags (around 75% precision and recall), we conclude that SSNs are effective in learning to parse. We anticipate that future developments using larger training sets, words as inputs, and a less constrained input-output format will make SSNs a real alternative to statistical methods.

3.1 SSNs for Parsing

The networks that are used in the experiments all have the same design. They all use the internal structure discussed in section 2.2 and illustrated in figure 2, and they all use the same input-output format. The input-output format is greatly simplified by SSNs' ability to represent constituents, but for these initial experiments some simplifying assumptions are still necessary. In particular, we want to define a single fixed input-output mapping for every sentence. This gives the network a stable pattern to learn, rather than having the network itself make choices such as when information should be output or which output constituent should be associated with which words. To achieve this we make

two assumptions, namely that outputs should occur as soon as theoretically possible, and that the head of each constituent is its first terminal child.

As shown in figure 2, SSNs have two sets of input units, constituent input units and gestalt input units. Defining a fixed input pattern for the gestalt inputs is straightforward, since these inputs pertain to information about the sentence as a whole. Whenever a tag is input to the network, the activation pattern for that tag is presented to the gestalt input units. The information from these tags is stored in the gestalt context units, forming a holistic representation of the preceding portion of the sentence. The use of this holistic representation is a significant distinction between SSNs and current symbolic statistical methods, giving SSNs some of the advantages of finite state methods. Figure 3 shows an example parse, and depicts the gestalt inputs as a tag just above its associated word. First NP is input to the gestalt component, then VVZ, then AT, and finally NN.

Defining a fixed input pattern for the constituent input units is more difficult, since the input must be independent of which tags are grouped together into constituents. For this we make use of the assumption that the first terminal child of every constituent is its head. When a tag is input we add a new constituent to the set of constituents that the network cycles through and assume that the input tag is the head of that constituent. The activation pattern for the tag is input in synchrony with this new constituent, but nothing is input to any of the old constituents. In the parse depicted in figure 3, these constituent inputs are shown as predications on new variables. First constituent w is introduced and given the input NP, then x is introduced and given VVZ, then y is introduced and given AT, and finally z is introduced and given NN.

Because the only input to a constituent is its head tag, the only thing that the constituent context units do is remember information about each constituent's first terminal child. This is not a very realistic assumption about the nature of the linguistic generalizations that the network needs to learn, but it is adequate for these initial experiments. This assumption simply means that more burden is placed on the network's gestalt memory, which can store information about any tag. Provided the appropriate constituent can be identified based on its first terminal child, this gestalt information can be transferred to the constituent through the combination units at the time when an output needs to be produced.

We also want to define a single fixed output pattern for each sentence. This is necessary since we use simple Backpropagation Through Time, plus it gives the network a stable mapping to learn. This desired output pattern is called the target pattern. The net-

Input	Output	Accumulated Output

NP(w)
NP
(John)

VVZ(x)
VVZ
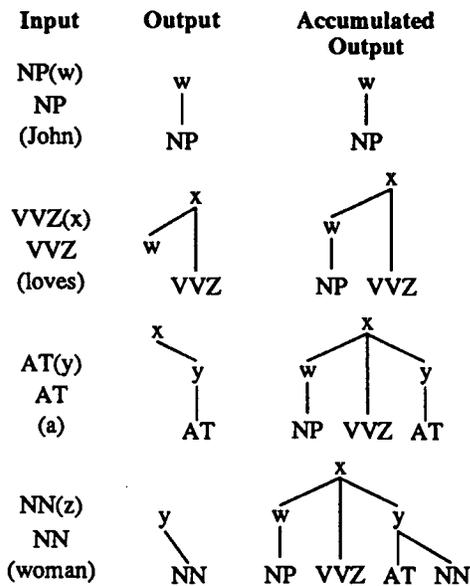(loves)

AT(y)
AT
(a)

NN(z)
NN
(woman)

Figure 3: A parse of "John loves a woman".

work is trained to try to produce this exact pattern, even though other patterns may be interpretable as the correct parse. To define a unique target output we need to specify which constituents in the corpus map to which constituents in the network, and at what point in the sentence each piece of information in the corpus needs to be output. The first problem is solved by the assumption that the first terminal child of a constituent is its head.[2] We map each constituent in the corpus to the constituent in the network that has the same head. Network constituents whose head is not the first terminal child of any corpus constituent are simply never mentioned in the output, as is true of z in figure 3. The second problem is solved by assuming that outputs should occur as soon as theoretically possible. As soon as all the constituents involved in a piece of information have been introduced into the network, that piece of information is required to be output. Although this means that there is no point at which the entire parse for a sentence is being output by the network, we can simply accumulate the network's incremental outputs and thereby interpret the output of the parser as a complete parse.

To specify an unlabeled parse tree it is sufficient to output the tree's set of parent-child relationships. For parent-child relationships that are between a constituent and a terminal, we know the constituent will have been introduced by the time the terminal's tag is input because a constituent is headed by its first terminal child. Thus this parent-child relationship should be output when the terminal's

---
[2] The cases where constituents in the corpus have no terminal children are discussed in the next subsection.

tag is input. This is done using a "parent" output unit, which is active in synchrony with the parent constituent when the terminal's tag is input. In figure 3, these parent outputs are shown structurally as parent-child relationships with the input tags. The first three tags all designate the constituents introduced with them as their parents, but the fourth tag (NN) designates the constituent introduced with the previous tag (y) as its parent.

For parent-child relationships that are between two nonterminal constituents, the earliest this information can be output is when the head of the second constituent is input. This is done using a "grandparent" output unit and a "sibling" output unit. The grandparent output unit is used when the child comes after the parent's head (i.e. right branching constituents like objects). In this case the grandparent output unit is active in synchrony with the parent constituent when the head of the child constituent is input. This is illustrated in the third row in figure 3, where AT is shown as having the grandparent x. The sibling output unit is used when the child precedes the parent's head (i.e. left branching constituents like subjects). In this case the sibling output unit is active in synchrony with the child constituent when the head of the parent constituent is input. This is illustrated in the second row in figure 3, where VVZ is shown as having the sibling w. These parent, grandparent, and sibling output units are sufficient to specify any of the parse trees that we require.

While training the networks requires having a unique target output, in testing we can allow any output pattern that is interpretable as the correct parse. Interpreting the output of the network has two stages. First, the continuous unit activations are mapped to discrete parent-child relationships. For this we simply take the maximums across competing parent outputs (for terminal's parents), and across competing grandparent and sibling outputs (for nonterminal's parents). Second, these parent-child relationships are mapped to their equivalent parse "tree". This process is illustrated in the rightmost column of figure 3, where the network's incremental output of parent-child relationships is accumulated to form a specification of the complete tree. This second stage may have some unexpected results (the constituents may be discontiguous, and the structure may not be connected), but it will always specify which words in the sentence each constituent includes. By defining each constituent purely in terms of what words it includes, we can compare the constituents identified in the network's output to the constituents in the corpus. As is standard, we report the percentage of the output constituents that are correct (precision), and percentage of the correct constituents that are output (recall).

535

3.2 A Corpus for SSNs

The Susanne[3] corpus is used in this paper as a source of preparsed sentences. The Susanne corpus consists of a subset of the Brown corpus, preparsed according to the Susanne classification scheme described in (Sampson, 1995). This data must be converted into a format suitable for the learning experiments described below. This section describes the conversion of the Susanne corpus sentences and the precision/recall evaluation functions.

We begin by describing the part of speech tags, which form the input to the network. The tags in the Susanne scheme are a detailed extension of the tags used in the Lancaster-Leeds Treebank (see Garside et al, 1987). For the experiments described below the simpler Lancaster-Leeds scheme is used. Each tag is a two or three letter sequence, e.g. 'John' would be encoded 'NP', the articles 'a' and 'the' are encoded 'AT', and verbs such as 'is' encoded 'VBZ'. These are input to the network by setting one bit in each of three banks of inputs; each bank representing one letter position, and the set bit indicating which letter or space occupies that position.

The network's output is an incremental representation of the unlabeled parse tree for the current sentence. The Susanne scheme uses a detailed classification of constituents, and some changes are necessary before the data can be used here. Firstly, the experiments in this paper are only concerned with parsing sentences, and so all constituents referring to the meta-sentence level have been discarded. Secondly, the Susanne scheme allows for 'ghost' markers. These elements are also discarded, as the 'ghost' elements do not affect the boundaries of the constituents present in the sentence.

Finally, it was noted in the previous subsection that the SSNs used for these learning experiments require every constituent to have at least one terminal child. There are very few constructions in the corpus that violate this constraint, but one of them is very common, namely the S-VP division. The linguistic head of the S (the verb) is within the VP, and thus the S often occurs without any tags as immediate children. For example, this occurs when S expands to simply NP VP. To address this problem, we collapse the S and VP into a single constituent, as is illustrtated in figure 3. The same is done for other such constructions, which include adjective, noun, determiner and prepositional phrases. This move is not linguistically unmotivated, since the result is equivalent to a form of dependency grammar (Melčuk, 1988), which have a long linguistic tradition. The constructions are also well defined enough

Expt	Training		Cross val		Test	
	Prec	Rec	Prec	Rec	Prec	Rec
(1)	75.6	79.1	66.7	71.9	60.4	66.5
(2)	71.6	75.8	68.2	73.8	62.6	69.4
(3)	64.2	71.4	58.6	66.9	59.8	68.5

Table 1: Results of experiments on Susanne corpus.

that the collapsed constituents could be separated at the interpretation stage, but we don't do that in these experiments. Also note that this limitation is introduced by a simplifying assumption, and is not inherent to the architecture.

3.3 Experimental Results

The experiments in this paper use one of the Susanne genres (genre A, press reportage) for the selection of training, cross-validation and test data. We describe three sets of experiments, training SSNs with the input-output format described in section 3.1. In each experiment, a variety of networks was trained, varying the number of units in the hidden and combination layers. Each network is trained using an extension of Backpropagation Through Time until the sum-squared error reaches a minimum. A cross-validation data set is used to choose the best networks, which are then given the test data, and precision/recall figures obtained.

For experiments (1) and (2), the first twelve files in Susanne genre A were used as a source for the training data, the next two for the cross-validation set (4700 words in 219 sentences, average length 21.56), and the final two for testing (4602 words in 176 sentences, average length 26.15).

For experiment (1), only sentences of length less than twenty words were used for training, resulting in a training set of 4683 words in 334 sentences. The precision and recall results for the best network can be seen in the first row of table 1. For experiment (2), a larger training set was used, containing sentences of length less than thirty words, resulting in a training set of 13,523 words in 696 sentences. We averaged the performance of the best two networks to obtain the figures in the second row of table 1.

For experiment (3), labeled parse trees were used as a target output. i.e. for each word we also output the label of its parent constituent. The output for the constituent labels uses one output unit for each of the 15 possible labels. For calculating the precision and recall results, the network must also output the correct label with the head of a constituent in order to count that constituent as correct. Further, this experiment uses data sets selected at random from the total set, rather than taking blocks from the corpus. Therefore, the cross-validation set in this case consists of 4551 words in 186 sentences, average length 24.47 words. The test set consists of

4485 words in 181 sentences, average length 24.78 words. As in experiment (2), we used a training set of sentences with less than 30 words, producing a set of 1079 sentences, 27,559 words. For this experiment none of the networks we tried converged to nontrivial solutions on the training set, but one network achieved reasonable performance before it collapsed to a trivial solution. The results for this network are shown in the third row of table 1.

From current corpus based statistical work on parsing, we know that sequences of part of speech tags contain enough information to achieve around 75% precision and recall on constituents (Charniak, forthcoming). On the other extreme, the simplistic parsing strategy of producing a purely right branching structure only achieves 34% precision and 61% recall on our test set. The fact that SSNs can achieve 63% precision and 69% recall using much smaller training sets than (Charniak, forthcoming) demonstrates that SSNs can be effective at learning the required generalizations from the data. While there is still room for improvement, we conclude that SSNs can learn to parse real natural language.

3.4 Extendability
The initial results reported above are very promising for future developments with Simple Synchrony Networks, as they are likely to improve in both the near and long term. Significant improvements are likely with larger training sets and longer training sentences. While other approaches typically use over a million words of training data, the largest training set we use is only 13,500 words. Also, fine tuning of the training methodology and architecture often improves network performance. For example we should be using larger networks, since our best results came from the largest networks we tried. Currently the biggest obstacle to exploring these alternatives is the long training times that are typical of Backpropagation Through Time, but there are a number of standard speedups which we will be trying.

Another source of possible improvements is to make the networks' input-output format more linguistically motivated. As an example, we retested the networks from experiment 2 above with a different mapping from the output of the network to constituents. If a word chooses an earlier word's constituent as its parent, then we treat these two words as being in the same constituent, even if the earlier word has itself chosen an even earlier word as its parent. 10% of the constituents are changed by this reinterpretation, with precision improving by 1.6% and recall worsening by 0.6%.

In the longer term the biggest improvement is likely to come from using words, instead of tags, as the input to the network. Currently all the best parsing systems use words, and back off to using tags for infrequent words (Charniak, forthcoming). Be-

cause connectionist networks automatically exhibit a frequency by regularity effect where infrequent cases are all pulled into the typical pattern, we would expect such backing off to be done automatically, and thus we would expect SSNs to perform well with words as inputs. The performance we have achieved with such small training sets supports this belief.

4   Conclusion
This paper demonstrates for the first time that a connectionist network can learn syntactic parsing. This improvement is the result of extending a standard architecture (Simple Recurrent Networks) with a technique for representing linguistic constituents (Temporal Synchrony Variable Binding). This extension allows Simple Synchrony Networks to generalize what they learn across constituents, thereby solving the sparse data problems of previous connectionist architectures. Initial experiments have empirically demonstrated this ability, and future extensions are likely to significantly improve on these results. We believe that the combination of this generalization ability with the adaptability of connectionist networks holds great promise for many areas of Computational Linguistics.

References

Eugene Charniak. forthcoming. Statistical techniques for natural language parsing. *AI Magazine*.

Jeffrey L. Elman. 1991. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7:195–225.

Jerry A. Fodor and Zenon W. Pylyshyn. 1988. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28:3–71.

R. Garside, G. Leech, and G. Sampson (eds). 1987. *The Computational Analysis of English: a corpus-based approach*. Longman Group UK Limited.

James Henderson. 1996. A connectionist architecture with inherent systematicity. In *Proceedings of the Eighteenth Conference of the Cognitive Science Society*, pages 574–579, La Jolla, CA.

I. Melčuk. 1988. *Dependency Syntax: Theory and Practice*. SUNY Press.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. 1986. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing, Vol 1*. MIT Press, Cambridge, MA.

Geoffrey Sampson. 1995. *English for the Computer*. Oxford University Press, Oxford, UK.

Lokendra Shastri and Venkat Ajjanagadde. 1993. 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.