Incremental Syntactic Parsing of Natural Language Corpora with Simple Synchrony Networks

Peter C.R. Lane and James B. Henderson

Abstract—This article explores the use of Simple Synchrony Networks (SSNs) for learning to parse English sentences drawn from a corpus of naturally occurring text. Parsing natural language sentences requires taking a sequence of words and outputting a hierarchical structure representing how those words fit together to form constituents. Feed-forward and Simple Recurrent Networks have had great difficulty with this task, in part because the number of relationships required to specify a structure is too large for the number of unit outputs they have available. SSNs have the representational power to output the necessary $O(n^2)$ possible structural relationships because SSNs extend the $O(n)$ incremental outputs of Simple Recurrent Networks with the $O(n)$ entity outputs provided by Temporal Synchrony Variable Binding. This article presents an incremental representation of constituent structures which allows SSNs to make effective use of both these dimensions. Experiments on learning to parse naturally occurring text show that this output format supports both effective representation and effective generalization in SSNs. To emphasize the importance of this generalization ability, this article also proposes a short-term memory mechanism for retaining a bounded number of constituents during parsing. This mechanism improves the $O(n^2)$ speed of the basic SSN architecture to linear time, but experiments confirm that the generalization ability of SSN networks is maintained.

Index Terms—Connectionist networks, natural language processing, simple synchrony networks, syntactic parsing, temporal synchrony variable binding.

1 Introduction

This article explores the use of Simple Synchrony Networks (SSNs) for learning to parse English sentences drawn from a corpus of naturally occurring text. The SSN has been defined in previous work [17], [23] and is an extension of the Simple Recurrent Network (SRN) [6], [7]. The SSN extends SRNs with Temporal Synchrony Variable Binding (TSVB) [33], which enables the SSN to represent structures and generalize across structural constituents.

We apply SSNs to syntactic parsing of natural language as it provides a standard task on real world data which requires a structured output. Parsing natural language sentences requires taking a sequence of words and outputting a hierarchical structure representing how those words fit together to form constituents, such as noun phrases and verb phrases. The state-of-the-art techniques for tackling this task are those from statistical language learning [2], [3], [5], [20]. The basic connectionist approach for learning language is based around the SRN, in which the network is trained to predict the next word in a sentence [7], [8], [30] or else trained to assess whether a sentence is grammatical or not [24], [25]. However, the simple SRN has not produced results comparable with the statistical parsers because its basic output representation is flat and unstructured.

The reason the simple SRN does not produce structured output representations lies with the required number of relationships which must be output to specify a structure such as a parse tree. For the SRN, only $O(n)$ relationships may be output, where n is the number of words in the sentence. However, a parse tree may specify a structural relationship between any word and any other word, requiring $O(n^2)$ outputs. This is not possible with the simple SRN because the length of a sentence is unbounded, but the number of output units is fixed. More fundamentally, even if a scheme is devised for bounding the required number of outputs to $O(n)$ (such as the STM mechanism discussed below), using large numbers of output units means learning a large number of distinct mappings and thus not generalizing across these distinctions. Thus, this article will focus not only on the representation of syntactic structures in the network's output, but crucially on a demonstration that the resulting networks generalize in an appropriate way when learning.

One example of a connectionist parser which uses multiple groups of output units to represent the multiple structural relationships is the Hebbian connectionist network in [13]. This network explicitly enforces generalization across structural constituents by requiring each group of output units to be trained on a random selection of the constituents. However, the amount of nontrainable internal structure required to enforce this generalization and

- *P.C.R. Lane is with ESRC Centre for Research in Development, Instruction, and Training, School of Psychology, University of Nottingham, University Park, Nottingham NG7 2RD, UK. E-mail: pcl@psychology.nottingham.ac.uk.*
- *J.B. Henderson is with the School of Engineering and Computer Science, Department of Computer Science, University of Exeter, Prince of Wales Road, Exeter EX4 4PT, UK. E-mail: j.b.henderson@exeter.ac.uk.*

Manuscript received 30 July 1999; revised 15 Feb. 2000; accepted 13 Mar. 2000.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 112696.

represent the possible forms of structure is a severe limitation. In particular, this component of the network would need to grow with the length and complexity of the sentences. This network has only been tested on a toy sublanguage and has not been demonstrated to scale up to the requirements of naturally occurring text.

The two alternatives to increasing the number of output units are to increase the amount of information represented by each unit's activation level and to increase the amount of time used to output the structure. Both these approaches are exemplified by the Confluent Preorder Parser [18]. As with other holistic parsers, the Confluent Preorder Parser first encodes the sentence into a distributed representation. This representation uses a bounded number of units, but the use of continuous activation values allows it to, in theory, encode a sentence of unbounded length. This distributed representation is then decoded into a different sequence which represents the sentence's syntactic structure (in particular, the preorder traversal of the structure). This output sequence is as long as it needs to be to output all the required structural relationships, thereby avoiding the restriction on the SRN that there can be only $O(n)$ outputs. But, both this decoding stage and the previous encoding stage miss important generalizations that are manifested in an explicitly structural representation and, thus, do not scale well to naturally occurring text. For the decoding stage, structural constituents which are close together in the structure may end up being far apart in the sequential encoding of that structure. Thus, important regularities about the relationship between these constituents will not be easily learned, while other regularities between constituents which just happen to occur next to each other in the sequence will be learned.[1] For the encoding stage, as sentences get longer, the ability of a fixed sized distributed representation to encode the entire sentence degrades. Indeed, [18] points out that the representational capacity of such approaches is limited, preventing their scale up beyond toy grammars.

Our approach is to represent structural constituents directly, rather than using one of the above indirect encodings. Thus, our connectionist architecture must be able to output the $O(n^2)$ structural relationships of a parse tree. To achieve this, the SSN extends the $O(n)$ incremental outputs of the SRN with the $O(n)$ entity outputs provided by TSVB. But, it is not enough to simply provide such a representation; the point of using a direct encoding is to enable the network to learn the regularities that motivated the use of a structured representation in the first place.[2] For the SSN, the use of TSVB means that learned regularities inherently generalize over structural constituents [15], [17], thereby capturing important linguistic properties such as systematicity [15]. It is this generalization ability which allows the SSN parser presented in this article to scale up to naturally occurring text.

In this article, we demonstrate how the SSN can represent the structured outputs necessary for natural language parsing in a way that allows the SSN to learn to parse from a corpus of real natural language text. To emphasize the generalization ability required to learn this task, we also introduce an extension to the SSN, namely, a short-term memory (STM) mechanism, which places a bound on the number of words which can be involved in any further structural relationships at any given time. This improves the $O(n^2)$ speed of the basic SSN architecture to linear time ($O(n)$), but it also means that only $O(n)$ relationships can be output. However, unlike previous connectionist approaches to parsing, this bound does not affect the ability of SSNs to generalize across this bounded dimension and, thus, to generalize in a linguistically appropriate way. Indeed, the performance of the SSN parser actually improves with the addition of the STM.

2 Simple Synchrony Networks

In this section, we provide a summary of Simple Synchrony Networks (SSNs) [17], [23]. We begin by describing the basic principles of Temporal Synchrony Variable Binding (TSVB) [33] which extend standard connectionist networks with pulsing units; pulsing units enable a network to provide output for each entity (word) encountered and not just for the current one, as with the standard connectionist unit. We briefly summarize the main equations defining the operation of TSVB networks and describe a training algorithm. Finally, we give three example SSN architectures; SSNs are defined by a restriction on the space of possible TSVB networks.

2.1 Trainable TSVB Networks

TSVB [33] is a connectionist technique for solving the "binding problem" through the use of synchrony of activation pulses. The binding problem arises where multiple entities are represented each with multiple properties; some mechanism is required to indicate which properties are bound to which entities. For example, on seeing two objects with the properties red, green, square, and triangle, some mechanism is needed to indicate which color relates to which shape. One method is to provide for variables x and y to stand for the two objects. The scene may then be unambiguously described as: $\mathrm{red}(x) \wedge \mathrm{green}(y) \wedge \mathrm{square}(x) \wedge \mathrm{triangle}(y)$. Another mechanism is to use synchrony binding, in which two units are representing properties bound to the same entity when they are pulsing synchronously. This proposal was originally made on biological grounds by Malsburg [36]. Earlier implementations of TSVB [14], [33] used nondifferentiable binary threshold units, and so could not be trained using standard connectionist techniques. In this section, we describe a different implementation of TSVB, one based on standard sigmoid activation units, which yields a trainable implementation of TSVB networks.

In order to implement a TSVB network, the central idea is to divide each time period into a number of *phases*; each phase will be associated with a unique entity. This

1. Methods, such as Long Short-Term Memory [19], can help learn regularities between distant items in a sequence, but they cannot totally overcome this unhelpful bias.

2. Note that this argument applies to any domain where structured representations have been found to be useful to express important regularities. Thus, although this article focuses on the requirements of parsing natural language, the SSN architecture would be relevant to any task which is best thought of as a mapping from a sequence of inputs to a structure.

correspondence of phases and entities means phases are analogous to variables; all units active in the same phase are representing information about the same entity, just as if the units were predicates on the same variable. Through the analogy with variables, the phase numbers play no role in determining unit activations within the network. There are two kinds of unit. The first is the *pulsing* unit, which computes in individual phases independent of other phases. The number of phases in each time period, $n(t)$, may vary, and so the pulsing unit's output activation is an $n(t)$-place vector, i.e., the activation, $o_j(\vec{t})$, of a pulsing unit j at time t is formed from $n(t)$ values, $\{o_j^p(t) \mid 1 \leq p \leq n(t)\}$, where $o_j^p(t)$ is the activation of unit j in phase p at time t. The second type of unit is the *nonpulsing* unit, which computes across all phases equally in the current time period; its output activation, $o_j(t)$, at time t is constant across every phase in time t.

We define the net input and output activation separately for each type of unit within a TSVB network based on the type of unit it is receiving activation from. We index the pulsing units in the network by a set of integers U_ρ and the nonpulsing units by a set of integers U_τ. Each noninput unit, j, receives activation from other units indexed by the set Inputs_j. Recurrent links are handled, without loss of generality, by adding context units to the network; each context unit's activation value is that which another unit had during the previous time step. The function, C, maps each unit to its associated context unit. Units are linked by real-valued weights: the link from unit i to unit j having the weight w_{ji}.

Given these definitions, the output activation of a pulsing unit, $j \in U_\rho$, in phase p at time t is defined as follows:

$$\mathrm{net}_j^p(t) = \sum_{i \in \mathrm{Inputs}_j} w_{ji} R^p(i,t)$$

$$\text{where} \quad R^p(i,t) = \begin{cases} o_i(t) & \text{if } i \in U_\tau \\ o_i^p(t) & \text{if } i \in U_\rho \end{cases}$$

$$o_j^p(t) = \begin{cases} \mathrm{in}_j^p(t) & \text{if } j \text{ is an input unit} \\ o_i^p(t-1) & \text{if } \exists i.j = C(i) \wedge t > 1 \\ 0 & \text{if } \exists i.j = C(i) \wedge t = 1 \\ \sigma(\mathrm{net}_j^p(t)) & \text{otherwise.} \end{cases}$$

With the standard sigmoid function: $\sigma(x) = 1/(1 + e^{-x})$. Note that the net function for each phase p takes activation from other pulsing units only in phase p, or from nonpulsing units, whose activation is the same across all phases. This is achieved by the function $R^p(i,t)$ which represents the activation of unit i in phase p at time t: nonpulsing units $(i \in U_\tau)$ have constant activation across each time period, so their activation is $o_i(t)$; pulsing units $(i \in U_\rho)$ output a separate activation for each phase of the time period, so their activation is $o_i^p(t)$.

The definition of the output activation of a nonpulsing unit is complicated by the possibility of a nonpulsing unit having inputs from pulsing units. In this case, activations from an unbounded number of phases would have to be combined into a single input value. As discussed in [22], including such links is not necessary and decreases the effectiveness of the architecture. Thus, they are not allowed in Simple Synchrony Networks. Given this simplification, the output activation of a nonpulsing unit, $j \in U_\tau$, at time t is defined as:

$$\mathrm{net}_j(t) = \sum_{i \in \mathrm{Inputs}_j} w_{ji} o_i(t)$$

$$o_j(t) = \begin{cases} \mathrm{in}_j(t) & \text{if } j \text{ is an input unit} \\ o_i(t-1) & \text{if } \exists i.j = C(i) \wedge t > 1 \\ 0 & \text{if } \exists i.j = C(i) \wedge t = 1 \\ \sigma(\mathrm{net}_j(t)) & \text{otherwise.} \end{cases}$$

Note that these nonpulsing units act just like a standard unit within an SRN.

In order to train these TSVB networks, we use a novel extension of Backpropagation Through Time (BPTT) [31]. When applying BPTT to a standard recurrent network, one copy of the network is made for each time step in the input sequence. Extending BPTT to TSVB networks requires a further copy of the network for every phase in the time period. The unfolding procedure copies both pulsing and nonpulsing units once per time period, and the pulsing units are copied additionally once per phase. As with standard BPTT, the unfolded network is a feed-forward network and can be trained using backpropagation. However, the unfolded network is a set of *copies* of the original and so, as training progresses, the changing weights must be constrained to ensure that each copy of a link uses the same weight; this is achieved by summing all the individual changes to each copy of the link.

2.2 SSN Architectures

Three example SSN architectures are illustrated in Fig. 1. Fig. 1 depicts layers of units as rectangles or blocks, each layer containing however many units the system designer chooses. Rectangles denote layers of nonpulsing units and blocks denote layers of pulsing units. Links between the layers (solid lines) indicate that every unit in the source layer is connected to every unit in the target layer. As discussed above, recurrence is implemented with context units, just as with SRNs [7] and the dotted lines indicate that activation from each unit in the source layer is copied to a corresponding context unit in the target layer.

All three architectures possess a layer of pulsing input units and a separate layer of nonpulsing input units. The procedure for inputting information to the SSNs is only a little different to that in standard connectionist networks. Consider a sequence of inputs "a b c...". A different pattern of activation is defined for each different input symbol, for example, activating one input unit to represent that symbol and having the rest of the input units inactive (i.e., a localist representation). With an SRN, the sequence of input patterns would be presented to the network in consecutive time periods. Thus, in time period 1, the SRN would receive the pattern for symbol "a" on its input; in time period 2, it would receive the pattern for symbol "b" on its input, etc. With the SSNs, the nonpulsing input units operate in just this way. The input symbol for each time period is simply presented on the nonpulsing units. For the pulsing units, each input symbol is introduced on a new phase, i.e., one unused by the input sequence to that point.

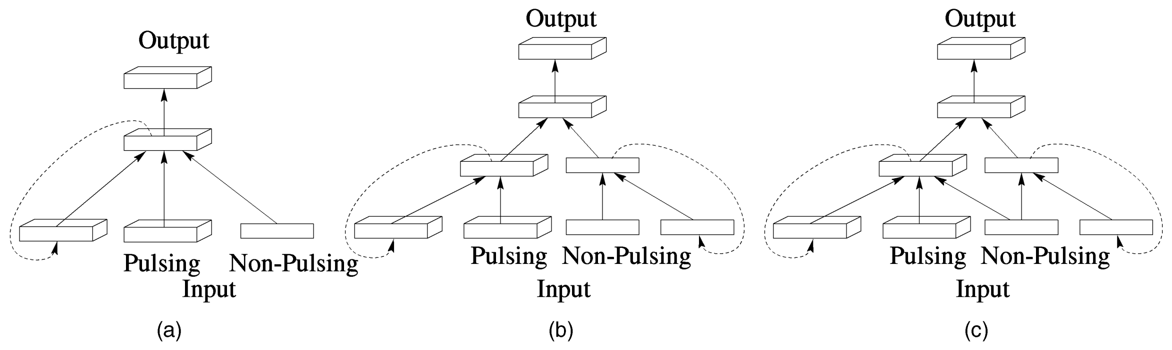

Fig. 1. Three Simple Synchrony Networks. Rectangles are layers of nonpulsing units and blocks are layers of pulsing units.

Thus, in time period 1, the SSN would receive the pattern for symbol "a" on its pulsing inputs in phase 1; in time period 2, the pattern for the symbol "b" would be presented on its pulsing inputs in phase 2; and so on.

The three architectures illustrated in Fig. 1 cover three different options for combining the information from the nonpulsing inputs with the information from the pulsing inputs. Essentially, the nonpulsing units contain information relevant to the sentence as a whole and the pulsing units information relevant to specific constituents. This information can be combined in three possible ways: before the recurrence (type A), after (type B), and both (type C). Given the constraint discussed in Section 2.1 that SSNs do not have links from pulsing to nonpulsing units, these three types partition the possible architectures. The combination layer in types B and C between the recurrent layers and the output layer is optional. We empirically compare the three illustrated architectures in the experiments in Section 4.

3 Syntactic Parsing

Syntactic parsing of natural language has been the center of a great deal of research because of its theoretical, cognitive, and practical importance. For our purposes, it provides a standard task on real world data which requires a structured output. A syntactic parser takes the words of a sentence and produces a hierarchical structure representing how those words fit together to form the constituents of that sentence. In this section, we discuss how this structure can be represented in a SSN and some of the implications of this representation.

3.1 Statistical Parsers

Traditionally, syntactic parsing was addressed by devising algorithms for enforcing syntactic grammars, which define what is and isn't a possible constituent structure for a sentence. More recent work has focused on how to incorporate probabilities into these grammars [2], [20] and how to estimate these probabilities from a corpus of naturally occurring text. The output structure is taken to be the structure with the highest probability according to the estimates. For example, probabilistic context-free grammars (PCFGs) [20] are context-free grammars with probabilities associated with each of the rewrite rules. The probability of an entire structure is the multiplication of the probabilities of all the rewrite rules used to derive that structure. This is a straightforward translation of context-free grammars into the statistical paradigm. The rule probabilities can be simply multiplied because they are assumed to be independent, just as the context-free assumption means the rules can be applied independently. Work in statistical parsing focuses on finding good independence assumptions, and it often takes as its starting point linguistic claims about the basic building blocks of a syntactic grammar.

Our SSN parser can be considered a statistical parser, in that the network itself is a form of statistical model. However, there are some clear differences. First, there is no "grammar" in the traditional sense. All the network's grammatical information is held implicitly in its pattern of link weights. More fundamentally, there are fewer independence assumptions. The network decides for itself what information to pay attention to and what to ignore. Statistical issues, such as combining multiple estimators or smoothing for sparse data, are handled by the network training.

But, as is usually the case with one-size-fits-all machine learning techniques, more domain knowledge has gone into the design of the SSN parser than is at first apparent. In particular, while very general, the input/output representation has been designed to make linguistic generalizations easy for the network to extract. For example, the SSN incrementally processes one word at a time and the output required at each time is related to that word. This not only reflects the incremental nature of human language processing, it also biases the network toward learning word-specific generalizations. The word-specific nature of linguistic generalizations is manifested in the current popularity of lexicalized grammar representations, as in [5]. Also, the short-term memory mechanism discussed later in this section is motivated by psycholinguistic phenomena. Other particular motivations will be discussed as they arise.

3.2 Structured Output Representations for SSNs

The syntactic structure of a natural language sentence is a hierarchical structure representing how the sentence's words fit together to form constituents, such as noun phrases and relative clauses. This structure is often specified in the form of a tree, with the constituents as nodes of the tree and parent-child relationships representing the hierarchy; all the words that are included in a child constituent are also included in its parent constituent. Thus, we can output a syntactic structure by outputting the set of constituents and the set of parent-child relationships between them and between them and the words.

The difficulty with outputting such a structure arises because of the number of parent-child relationships. On linguistic grounds, it is safe to assume that the number of constituents is linear in the number of words.[3] Thus, we can introduce a bounded number of entities with each word and have one entity for each constituent.[4] The problem is that this still leaves $O(n^2)$ (quadratic) possible parent-child relationships between these $O(n)$ (linear) constituents. The solution is to make full use of both the $O(n)$ times at which words are presented to the network and the $O(n)$ entities at any given time. Thus, we cannot wait until all the words have been input and then output the entire structure, as is done with a symbolic parser. Instead, we must incrementally output pieces of the structure such that by the time the whole sentence has been input, the whole structure has been output.

The first aspect of this solution is that new constituents are introduced to the SSN with each word that is input. In other words, during each period new phases (i.e., entities) must be added to the set of phases that are being processed. As illustrated in Fig. 1, SSNs have two banks of input units, pulsing and nonpulsing. The nonpulsing input units hold the part of speech tag for the word being input during the current period. (For simplicity, we do not use the words themselves.) Using the pulsing input units, this word-tag is also input to the new phase(s) introduced in the current period. In this way, the number of constituents represented by the network grows linearly with the number of words that have been input to it. For the experiments discussed below, we make a slightly stronger assumption (for reasons detailed below), namely, that only one constituent is added with each word-tag. This means that the network can only output parse trees which contain at most as many constituents as there are words in the sentence. This simplification requires some adjustments to be made to any preparsed corpus of naturally occurring text, but is not linguistically unmotivated; the result is equivalent to a form of dependency grammar [27] and such grammars have a long linguistic tradition. We will return to the adjustments required to the training set when considering the corpus used in the experiments in Section 4.

Now that we have $O(n)$ constituents, we need to ensure that enough information about the structure is output during each period so that the entire structure has been specified by the end of the sentence. For this, we need to make use of the pulsing output units illustrated in Fig. 1. Our description of the parsing process refers to the example in Fig. 2, which illustrates the sentence "Mary saw the game was bad." This sentence is represented as a sequence of word-tags as "NP VVD AT NN VBD JJ" and the sentence structure (S) contains separate constituents for the subject noun (N) and object clause (F), which contains a further

3. This simply means that there aren't unbounded numbers of constituents that contain the same set of words (unbranching constituents).

4. Alternatively, we could introduce one entity with each word and have one entity represent a bounded number of constituents. As will be discussed below, we are assuming that the bound is 1, so these two alternatives are equivalent.

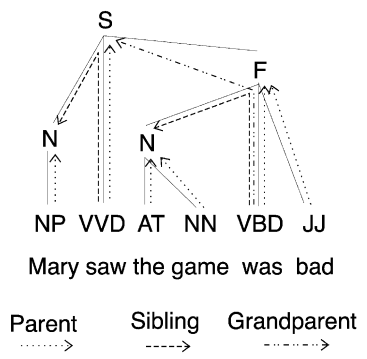

Fig. 2. A sample parse tree. The solid lines indicate the parse tree itself, the dotted and dashed lines the relationship between the words and nodes.

noun phrase (N). Note that this parse tree has had some constituents conflated to comply with the constraint that there be only one constituent per word; its relation to standard parse-tree representations is covered in Section 4.

The pulsing units in the network, during the *n*th time period, provide an output for each of the n constituents represented by the SSN at that period. As mentioned above, we obviously want these outputs to relate to the *n*th word-tag, which is being input during that period. So, one thing we want to output at this time is the parent of that word-tag within the constituent structure. Thus, we simply have a *parent* output unit, which pulses during each period in the phase of the constituent which is the parent of the period's word-tag. Examples of these parent relationships are shown in Fig. 2, and examples of these outputs are shown in Table 1. For the experiments discussed below, we assume that each constituent is identified by the first word-tag which attaches to it in this way. So, if this is the first word-tag to attach to a given constituent, then its parent is the constituent introduced with that word-tag, as for the NP, AT, VVD, and VBD in the example. Otherwise, the word-tag's parent is the constituent introduced with its leftmost sibling word, as for the NN and JJ in the example. In these cases, the newly introduced constituent simply does not play any role in the constituent structure.

The *parent* output unit is enough to specify all parent-child relationships between constituents and word-tags, leaving only the parent-child relationships between constituents. For these, we take a maximally incremental approach; such a parent-child relationship needs to be output as soon as both the constituents involved have been introduced into the SSN. There are two such cases, when the parent is introduced before the child and when the child is introduced before the parent. The first case is covered by adding a *grandparent* output unit. This output specifies the grandparent constituent for the current word-tag, as illustrated in Fig. 2 and Table 1 for the VBD. This grandparent constituent must be the parent of the constituent which is the parent of the current word-tag. In other words, the combination of the *parent* output and the *grandparent* output specifies a parent-child relationship between the word-tag's grandparent and its parent. The second case is covered similarly by a *sibling* output unit. This output specifies any constituent which shares the same parent as the current word-tag, as illustrated in Fig. 2 and

TABLE 1
The Input Information and Corresponding Grandparent/Parent/Sibling Outputs for the Sample Structure Shown in Fig. 2

Time Period	Pulsing Inputs Phase						Non-pulsing Inputs	Structure Outputs Phase					
	1	2	3	4	5	6		1	2	3	4	5	6
1	NP						NP	P					
2		VVD					VVD	S	P				
3			AT				AT			P			
4				NN			NN			P			
5					VBD		VBD		G	S		P	
6						JJ	JJ					P	

Table 1 for the VVD and VBD. The combination of the *parent* output and the *sibling* output specifies a parent-child relationship between the word-tag's parent and each of its siblings (possibly more than one). In the experiments below, the *grandparent* and *sibling* outputs are only used during the period in which the word-tag's parent is first introduced.

The interpretation of the outputs is best described with reference to a detailed trace of activation on the input and output units, as provided in Table 1 for the structure in Fig. 2. The first two columns of the table (other than the period numbers) show the inputs to the SSN. The first column shows the activation presented to the bank of pulsing input units. Each row of the column represents a separate time period, and the column is divided into separate phases, six being sufficient for this example. Each cell indicates the information input to the network in the indicated time period and phase; the word-tag appearing in the cell indicates which of the pulsing input units is active in that time period and phase. The second column shows the activation presented to the bank of nonpulsing input units. No phase information is relevant to these units, and so the word-tag given simply indicates which of the nonpulsing input units is active in each time period.

The third column in the table shows the activation present on the output units. In every period, the *parent* (P) output is active in the phase for the constituent which the period's word-tag is attached to. In period 1, an NP (proper noun) is input and this is attached to constituent number 1. This is the constituent which was introduced in period 1 and it is specified as the word-tag's parent because no previous word-tags have the same parent (there being no previous word-tags in this case). The same thing happens with the *parent* outputs in periods 2, 3, and 5. In period 4, the word-tag NN (noun) is input and attached to constituent 3, which is the constituent which was introduced during the input of AT (article) in the previous period. This specifies that this AT and NN have the same parent, namely, constituent 3. The same relationship is specified in period 6 between the JJ (adverb/adjective) being input and the preceding VBD (verb).

Given the set of constituents identified by the *parent* outputs, the *grandparent* (G) and *sibling* (S) outputs specify the structural relationships between these constituents. Because we only specify these outputs during the periods in which a new constituent has been identified, only periods 2, 3, and 5 can have these outputs (period 1 has no other constituents to specify relationships with). In period 2, constituent 1 is specified as the sibling of the VVD word-tag being input. Thus, since constituent 2 is the parent of VVD, constituent 2 must also be the parent of constituent 1. Constituent 2 does not itself have a parent because it is the root of the tree structure. In period 3, no siblings or grandparent are specified because constituent 3 has no constituent children and its parent has not yet been introduced. In period 5, the parent of constituent 3 is specified as constituent 5 through the *sibling* output, as above. Also, in period 5, the *grandparent* output unit pulses in phase 2. This specifies that constituent 2 is the grandparent of the VBD word-tag being input in period 5. Thus, constituent 2 is the parent of constituent 5 since constituent 5 is the parent of VBD. Note that the use of phases to represent constituents means that no special mechanisms are necessary to handle this case, where one constituent (F) is embedded within another (S).

In addition to structural relationships, natural language syntactic structures also typically include labels on the constituents. This is relatively straightforward for SSNs to achieve. We use an additional set of pulsing output units, one for each label. The network indicates the label for a given constituent by pulsing the label's unit in phase with the new constituent when it is introduced to the parse tree.

So far, we have described the target output for a SSN, in which units are either pulsing or not. Being based on SRNs, the actual unit outputs are, of course, continuous values between 0 and 1. There are a variety of ways to interpret these patterns of continuous values as a specification of constituency. In the experiments discussed below, we simply threshold them; all units with activations above 0.6 are treated as "on." The indicated set of structural relationships is then converted to a set of constituents, which may then be evaluated using the precision and recall measures standard in statistical language learning; precision is the percentage of output constituents that are correct and recall the percentage of correct constituents that are output.

The important characteristic of SSNs for outputting structures is that just three output units, *grandparent*, *parent*, and *sibling*, are sufficient to specify all the structures allowed by our assumptions that at most one constituent needs to be introduced with each word.[5] We call this the

5. As mentioned above, the set of allowable structures can be expanded either by increasing the number of entities introduced with each word or by expanding the number of structural relationships so as to allow each entity to represent more than one constituent. In either case, the number of constituents still must be linear in the number of words. This constraint could only be relaxed by allowing unbounded computation steps per word input.

GPS representation. As the SSN proceeds incrementally through the sentence, at each word-tag, it outputs the word-tag's parent and (if it is the parent's first word-child) all the parent-child relationships between that parent and the previous constituents in the sentence. By the time the SSN reaches the last word of the sentence, its cumulative output will specify all the parent-child relationships between all the constituents, thus specifying the entire hierarchical structure of the sentence's constituency.

3.3 Inherent Generalizations

In Section 2, we described the SSN, its training algorithm, and a variety of possible SSN architectures. Because the definition of TSVB units retains and augments the properties of standard feed-forward and recurrent connectionist networks, SSNs retain the advantages of distributed representations and the ability to generalize across sequences. The SSN has also been shown to support a class of structured representation. Although this representation in itself is important in extending the range of domains to which connectionist networks may be applied, the SSN's use of phases to identify structural constituents also confers a powerful generalization ability, specifically, the ability to generalize learned information across structural constituents.

As an example of this kind of generalization, consider what is required to generalize from the sentence "John loves Mary" to the sentence "Mary loves John." In both sentences, the network needs to output that "John" and "Mary" are noun phrases, that the noun phrase preceding the verb is the subject, and that the noun phrase after the verb is the object. In order to generalize, it must learn these four things independently of each other and yet, for any particular sentence, it must represent the binding between each constituent's word and its syntactic position.

The SSN achieves this generalization ability by using temporal phases to represent these bindings, but using link weights to represent these generalizations. Because the same link weights are used for each phase, the information learned in one phase will inherently be generalized to constituents in other phases. Thus, once the network has learned that the input "Mary" correlates with being a noun phrase, it will produce a noun phrase output regardless of what other features (such as syntactic position) are bound to the same phase. Similarly, the network will learn that a noun phrase preceding a verb correlates with the noun phrase being the subject of the verb, regardless of what other features (such as the word "Mary") are bound to the same phase. Then, even if the network has never seen "Mary" as a subject before, the application of these two independent rules in the same phase will produce a pattern of synchronous activation that represents that "Mary" is the subject. Henderson [15] has shown how this inherent ability of TSVB networks to generalize across constituents relates to systematicity [9].

3.4 Short-Term Memory

In learning-based systems, it is the system's ability to generalize from training sets to testing sets that determines its value. This implies that the real value of the SSN is in its ability to generalize over constituents, and not its ability to output $O(n^2)$ structural relationships. This suspicion is confirmed when we consider some specific characteristics of our domain, natural language sentences. It has long been known that constraints on people's ability to process language put a bound on constructions, such as center embedding [4], which are the only constructions that would actually require allowances for $O(n^2)$ structural relationships. For example, "the rat that the cat that the dog chased bit died" is almost impossible to understand without pencil and paper, but "the dog chased the cat that bit the rat that died" is easy to understand. The basic reason for this difference is that, in the first case, all the noun phrases need to be kept in memory so that their relationships to the later verbs can be determined, while in the second case, each noun phrase can be forgotten as soon as the verb following it has been seen.

Motivated by this observation and by work showing that, in many other domains, people can only keep a small number of things active in memory at any one time [28], we have added a "short-term memory" (STM) mechanism to the basic SSN architecture. This mechanism improves the SSN's efficiency to $O(n)$ time. The definition of the SSN so far has stated that each word-tag input to the network will be input into a new phase of the network. Information is then computed for all of these phases in every subsequent time period. However, the bound on the depth of center embedding implies that, in any given time period, only a relatively small number of these phases will be referred to by later parts of the parse tree. The trick is to work out which of the phases are going to be relevant to later processing and only compute information for these phases. The idea we use here is a simple one based on the idea of the audio-loop proposed by Baddeley [1].

Instead of computing all phases in the current time period, we instead compute only those in a STM queue. This queue has a maximum size, which is the bound on STM referred to above. When a new phase is introduced to the network, this phase is added to the head of the queue. When a phase is referred to by one of the output units, that phase is moved to the head of the queue. This simple mechanism means that unimportant phases, i.e., those which are not referred to in the output, will move to the end of the queue and be forgotten. Note that, in training, the target outputs are used to determine which phases are moved to the head of the queue and not the actual outputs, thereby ensuring that the network learns only about the relevant phases. Also, information held by the network about word-tags and constituents are specific to *phases*, not position in STM or the input word order. Therefore, items cannot be confused during the reordering process which occurs in the STM. Indeed, it is precisely this use of phases for representing constituents which allows the SSN to keep its ability to generalize over constituents and still be able to parse in $O(n)$ time.

4 Experiments in Syntactic Parsing with SSNs

In this section, we describe some experiments training a range of SSNs to parse sentences drawn from a corpus of real natural language. The experiments demonstrate how the SSN may be used for connectionist language learning with structured output representations. Also, the fact that the SSN's performance is evaluated in terms of precision and recall of constituents means that the SSN's performance may be directly compared with statistical parsers. We first describe the corpus used, provide some results, and then give some analysis of the training data.

4.1 A Natural Language Corpus

We use the SUSANNE corpus as a source of preparsed sentences for our experiments. The SUSANNE corpus is a subset of the Brown corpus, and is preparsed according to the SUSANNE classification scheme described in [32]. In order to use the SUSANNE corpus, we convert the provided information into a format suitable for presentation to our parser.

The SUSANNE scheme provides detailed information about every word and symbol within the corpus. We use the word-tags as input to the network, due to time constraints and the limited size of the corpus. The word-tags in SUSANNE are a detailed extension of the tags used in the Lancaster-Leeds Treebank [12]. In our experiments, the simpler Lancaster-Leeds scheme is used. Each word-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" are encoded "VBZ." Each word-tag is input to the network by setting one bit in each of three banks of input; each bank representing one letter position, and the set bit indicating which letter or space occupies that position. In order to construct the set of constituents forming the target parse tree, we first need to extract the syntactic constituents from the wealth of information provided by the SUSANNE classification scheme. This includes information at the metasentence level, which can be discarded, and semantic relations between constituents.

Finally, as described above, the GPS representation used in our experiments requires that every constituent have at least one terminal child. This limitation is violated by few constructions, though one of them, the S–VP division, is very common. For example, in the sentence "Mary loves John," a typical encoding would be: [S [NP Mary] [VP [V loves] [NP John]]]. The linguistic head of the S (the verb "loves") is within the VP and, so, the S does not have any tags as immediate children. To address this problem, we collapse the S and VP into a single constituent, producing: [S [NP Mary] [V loves] [NP John]]. The same is done for other such constructions, which include adjective, noun, determiner, and prepositional phrases. With the corpus used here, the number of changes is fairly minor.[6]

4.2 Results

One of the SUSANNE genres (genre A for press reportage) was chosen for the experiments and training, cross-validation, and test sets were selected at random from the total in the ratio 4:1:1. Sentences of fewer than 15 word-tags were selected from the training set to form a set of 265 sentences containing 2,834 word-tags, an average sentence length of 10.7. Similarly, a cross-validation set was selected, containing 38 sentences of 418 word-tags, average sentence length of 11.0, and a test set with 34 sentences, containing 346 words, with an average sentence length of 10.2.

The three SSN types A, B, and C were all tested. Twelve networks were trained from each type, consisting of four sizes of network (between 20 and 100 units in each layer), each size was tested with three different STM lengths (3, 6, and 10). Each network was trained on the training set for 100 epochs, using a constant learning rate η of 0.05.

Table 2 gives figures for five networks. For each network, the performance on the three data sets (training, cross-validation, and test) are given under three categories: the number of correct sentences, a measure of the number of correct constituents (precision and recall), and the percentage of correct responses on each output unit. The measure of precision and recall used for constituent evaluation is a standard measure used in statistical language learning [20]. The *precision* is the number of correct constituents output by the parser divided by the total number of constituents output by the parser. The *recall* is the number of correct constituents divided by the number in the target parse. Each constituent is counted as correct if it contains the same set of words as the target and has the same label.[7] The presence of this measure in our results is significant because it confirms the similarity of the input-output representations used by the SSN with those used by statistical parsers and, therefore, some direct comparisons can be made; we return to this point in Section 5.

Considering the figures in Table 2, the type A networks are not particularly successful, with only the rare sentence being correctly parsed. Results for the best performing type A network are given in the first row of the table. The type B and C networks were much more successful. For each type, the results from two networks are given: the first having the best average precision/recall measure and the second having better results on the individual outputs (i.e., G, P, S, and label). Both the type B and C networks produce similar ranges of performance: around 25 percent of sentences are correct and between 70 and 80 percent are scored in average precision/recall by the better networks. In particular, the percentage correct for the constituent labels and the P output exceed 90 percent. Also notable is that the percentage results of the networks are similar across the three data sets, indicating that the network has learned a robust mapping from input sentences to output parse trees. This level of generalization (around 80 percent average precision/recall) is similar to that achieved by PCFG parsers [20], although, for a fair comparison, identical experiments must be performed with each algorithm; again, we return to this point in Section 5.

4.3 Analysis of Results

The basic experimental results above have provided both detailed values of the performance of the network with

6. Out of 1,580 constituents, 265 have been lost to the S–VP change, 28 similar changes were made to relative clauses, and only 12 adjustments were required to other nonverb constructions—most of the verb clauses could be artificially reintroduced on output, which leaves around 30 irrecoverable changes to the corpus structure.

7. Precision may be compared with the standard measure of "errors of commission," and recall with the standard "error of omission."

TABLE 2
Comparison of Network Types in Learning to Parse

Test	Sentences	Precision	Recall	G	P	S	Label
Type A : STM of 10, 100 units							
Train	1/265 (0%)	34.7	31.3	35.6	65.9	20.8	89.3
Cross	0/38 (0%)	31.8	29.7	30.5	65.2	18.7	84.7
Type B : STM of 3, 80 units in each layer							
Train	63/265 (24%)	69.9	68.7	74.1	94.3	73.0	97.5
Cross	10/38 (26%)	71.5	71.2	74.8	95.6	70.7	96.4
Test	9/34 (25%)	71.2	70.8	82.0	94.6	64.6	98.3
Type B : STM of 6, 80 units in each layer							
Train	62/265 (24%)	66.0	64.4	75.2	92.3	84.6	97.5
Cross	12/38 (32%)	65.3	64.2	82.4	92.6	85.3	96.2
Test	8/34 (25%)	67.4	65.2	83.0	93.3	83.1	98.3
Type C : STM of 3, 80 units in each layer							
Train	64/265 (24%)	72.6	71.7	70.8	95.7	72.1	98.4
Cross	9/38 (24%)	74.4	73.8	71.8	97.3	70.7	97.8
Test	13/34 (38%)	80.3	80.3	85.0	96.0	66.1	99.1
Type C : STM of 6, 80 units in each layer							
Train	56/265 (15%)	65.5	63.7	68.1	92.8	82.3	97.1
Cross	8/38 (21%)	63.6	61.1	68.7	91.5	81.3	95.5
Test	10/34 (29%)	71.4	70.2	76.0	93.9	84.7	98.3

specific output relationships as well as their combined performance in terms of the constituent-level measures of precision and recall. Here, these results are broken down and compared to see the progress of learning the networks over time with a comparison of the effect of the STM queue, and a table of the actual dependencies present in the data itself.

4.3.1 Effects of STM Length

The effects of STM length can be seen by plotting the performance of one type of network with varying sizes of STM. This is done in Fig. 3, in which the performance of a type C network with 80 units in every hidden layer is shown for the three sizes of STM, i.e., 3, 6, and 10. The separate graphs show the constituent-level performance of the network, in terms of average precision/recall, and the performance of the separate output units, grandparent, parent, sibling, and constituent label (this latter, though a group of units, is treated as a single output). The graphs demonstrate that, at the constituent-level, the shorter STM lengths perform better. However, the longer STM lengths can achieve greater accuracy with specific outputs, in particular, with respect to the sibling output. This is to be expected, as the longer lengths preserve more information and, so, have a greater likelihood of containing the phase referred to by the specific output.

4.3.2 Dependency Lengths in the Data Set

An important concern in connectionist language learning has been the length of dependency which the SRN can learn [8]. In this section, we provide an analysis of our data set to see exactly what dependency lengths are present in a corpus of naturally occurring text.

Table 3 contains an analysis of the lengths of each dependency contained in sentences with a maximum length of 30 words. The length of a dependency is the number of words between the current word and that indicated by each output. The table separately lists the lengths for each of the output units, with the final two columns providing a total number and percentage for that dependency length across the whole corpus. The surprising result of this table is that most of the dependencies (almost 70 percent) relate to the current word or its predecessor. There is a sharp tailing off of frequency as we consider longer dependencies—Table 3 only shows the shortest lengths; the lengths tail off gradually to a length of 25 words.

With the STM, the network can only process a limited number of words at any one time and, so, the length of dependency which the network can handle is altered. With a STM, the length of dependency will be the number of places down the queue which each phase has progressed before being required. So, in Table 4, we provide a similar analysis to that above, but this time, instead of counting the length as the number of intervening phases, we count the length of each dependency as the position which that phase occupies in the STM. Thus, if a phase is in the third position of the STM when it is required, then the length of the dependency will be given as three. Table 4 shows similar effect in the range of dependency lengths to that in Table 3, although there is a greater concentration in the shortest lengths, as desired. The limited number of longer dependencies (not shown) still extend to length 25; these are isolated words or punctuation symbols which are referred to only once.

4.3.3 Conclusions on Experiments

The impact of the STM is quite considerable in respect to training times, reducing them by at least an order of magnitude. As discussed above, the actual lengths of dependencies encountered by the network are not changed much by the addition of a STM. The experiments show that longer STMs achieve better performance on some specific

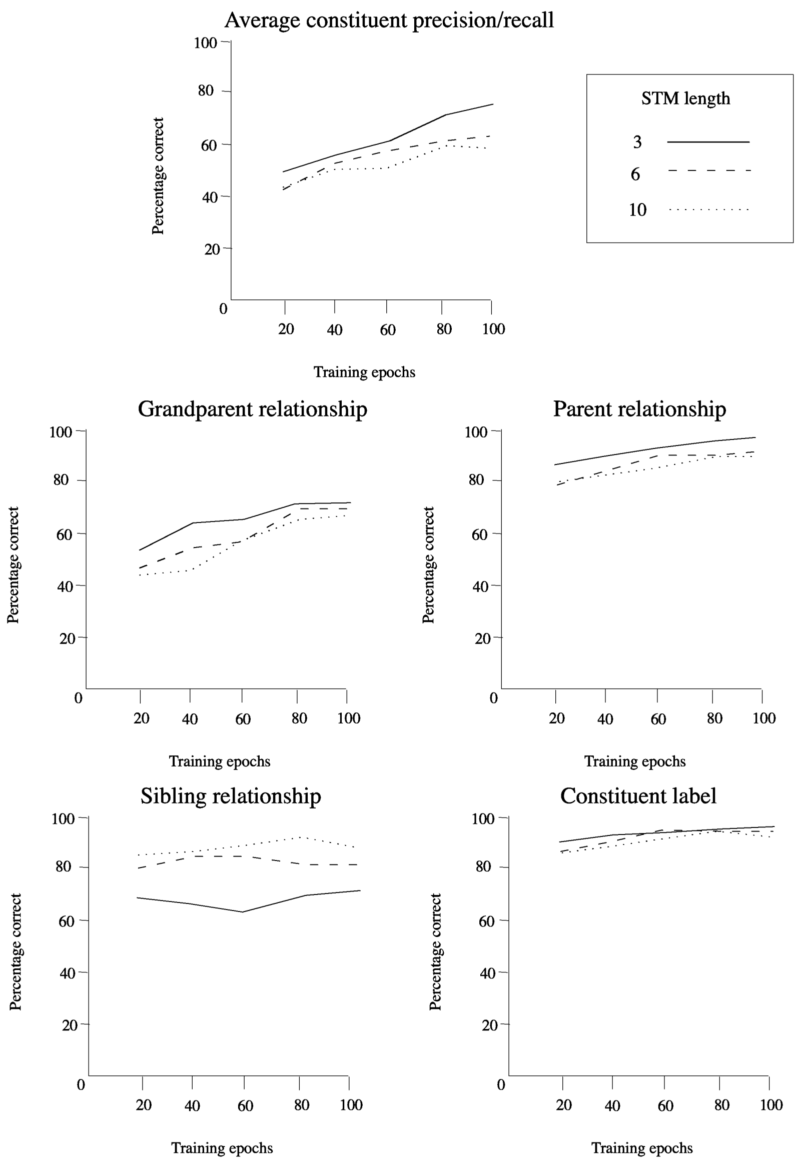

Fig. 3. Comparison of the effects of STM length on a type C network with 80 units in every hidden layer.

outputs of the network; however, the shorter STM still yields the best level of constituent accuracy. This difference is of interest, as the choice of STM length depends on one's measure of performance. A better performance is achieved on specific outputs with a longer STM because your desired output is more likely to appear in the STM. But, a better performance is achieved at the constituent level, based on a competition between different outputs because the smaller STM reduces the likelihood of spurious outputs competing with the correct ones. Also note the domain specificity of this last point: The smaller STM only works because natural language itself has a bias toward shorter dependencies.

TABLE 3
Dependencies by Type and length in Sentences with Fewer than 30 Words

Dependency Length	G	P	S	total	%=length	%<length
0	0	7354	0	7354	38.0%	38.0%
1	2268	3134	763	6165	31.8%	69.8%
2	1018	1037	502	2557	13.2%	83.0%
3	568	350	211	1129	5.8%	88.8%
4	351	173	106	630	3.3%	92.0%
5	225	101	78	404	2.1%	94.1%
Mean length	2.7	0.8	3.3	1.6		

Not all dependencies are shown, the greatest length is 25 words. Number of sentences: 716, number of words: 13,472, and average length: 18.8.

TABLE 4
Dependencies by Type and Length Across the STM for the Sentences from Table 3

STM Dependency Length	G	P	S	total	%=length	%<length
0	0	7354	0	7354	38.0%	38.0%
1	2614	3011	1128	6753	34.9%	72.8%
2	1394	1513	310	3217	16.6%	89.4%
3	397	252	150	799	4.1%	93.5%
4	284	127	78	489	2.5%	96.1%
5	135	42	68	245	1.3%	97.3%
Mean length	2.0	0.7	2.7	1.2		

5 Discussion

This article has focused on how SSNs can use an incremental representation of constituent structure in order to learn to parse. In addition, we have shown that arguments about the generalization abilities necessary to learn to parse are distinct from arguments about bounds on those abilities through the introduction of the STM mechanism, which enhances the efficiency of the basic SSN without harming its ability to learn to parse. In this section, we consider in brief the importance of these results for connectionist language learning and how our model compares with other extensions to SRNs for handling structured representations.

First, the experiments described above demonstrate how a connectionist network can successfully learn to generate parse trees for sentences drawn from a corpus of naturally occurring text. This is a standard task in computational language learning using statistical methods. Because the same performance measures (precision/recall) can be applied to the output of the SSN as with a typical statistical method, such as the simple Probabilistic Context Free Grammar (PCFG), direct comparisons can be made between the two approaches. For instance, the simple PCFG can achieve around 72 percent average precision/recall [20] on parsing from sequences of word-tags. In comparison, the SSN in the above experiments achieves 80 percent average precision/recall when trained and tested on sentences with fewer than 15 words. However, this is not a fair comparison, as the corpora sizes and contents are dissimilar.

In an extension to the work here, Henderson [16] has presented a slight variant of the basic SSN model and compared its performance directly with that of PCFGs on identical corpora. In those results, the PCFG, due to the restricted size of the training set, was only able to parse half the test sentences with a precision/recall figure of 54 percent/29 percent. In comparison, the SSN was able to parse all the sentences and yielded a performance of 65 percent/65 percent. Even if we only count the parsed sentences, the PCFG only had a performance of 54 percent/58 percent, compared to the SSN's performance of 68 percent/67 percent on that subset. The variations introduced by Henderson [16] to the SSN mostly affect the input layer. In this article, the pulsing inputs to the SSN receive input only for newly introduced phases, requiring the network to remember the previous periods' input. In [16], the pulsing input from the previous period is carried forward in its particular phase. An additional pulsing input unit is then used to distinguish the newly introduced phase from the others. Because of this change in input representation, the results in [16] have been achieved with a type A SSN.

As noted in Section 1, experiments with natural language using SRNs have typically used a restricted form of input representation, either predicting the next word in a sentence [6], [8], [30] or assessing whether it is grammatical [24], [25]. Our extension to the SRN, the SSN, corrects this limitation by enhancing the range of output representations to include structured parse trees. Our approach is designed to *generate* a structured representation given a sequence of input data. The generation aspect of this task largely distinguishes our approach from other extensions to SRNs for handling structured data. For example, the Backpropagation Through Structure (BPTS) algorithm [35], [11] assumes that the network is being trained to *process* structured input data, either for classification [10] or for transformation [11]. The transformation task is closer to that of training a parser, but, as the conclusion of [11] makes clear, the use of BPTS relies on the input and output having the same structural form, which prevents such networks being directly applicable to the task of generating a parse tree from a sequence of input word-tags. However, there is a relationship between the BPTS and the SSN in terms of the SSN's temporal structure. The SSN is trained using an extension of Backpropagation Through Time, where the network is unfolded over its two temporal dimensions, period, and phase. This is one specific instance of BPTS, where the structure in question is the temporal structure. However, the mapping from this temporal structure to the structures in the domain is done as part of the interpretation of the network's output activations (using the GPS representation), and thus does not fall within the BPTS framework. In the broader context of transforming structured data, the SSN and the incremental parse tree representation described in this article thereby offer one way for a connectionist network to generate structured output data from an unstructured input.

Apart from models based on SRNs, other forms of connectionist network have been proposed for handling the types of structured information required for language learning. For instance, Hadley and Hayward [13] propose a highly structured network which learns to generalize across syntactic structures in accordance with systematicity [9]. However, this approach is limited due to the amount of nontrainable internal structure required to enforce the appropriate generalizations. As discussed in Section 3.3, (and Henderson [15]), the SSN relies on temporal synchrony to produce a similar effect, which renders the generalization ability of SSNs largely independent of its specific architectural details. Indeed, in experiments training a type B SSN on the same recursive grammar to that of Hadley and Hayward [13], a similar ability to generalize across syntactic structures was demonstrated [21], [22].

As the above discussion makes clear, the *identical* SSN network learns effectively with both a specific toy grammar and a corpus of naturally occurring text. This is because the added ability to generalize over constituents allows the SSN to generalize in a more linguistically appropriate way, and thus deal with the high variability in naturally occurring text. This transfer from a toy domain to a real corpus of sentences sets the SSN apart from a number of other proposals for connectionist language learning, which tend to be limited to applications involving toy grammars alone. These include the approaches that encode sentences recursively into a distributed representation, such as holistic parsers [18] or labeled-RAAMs [34]. The number of cycles in this recursive encoding depends on the size of the parse tree, which means that the performance degrades as the complexity of the sentences increases. This makes it difficult to apply these approaches to naturally occurring text. In our SSN, we address this specific problem by not attempting to encode everything into a distributed representation prior to extracting the parse tree, but by incrementally outputting pieces of the parse tree. This incremental approach to parsing presents a different model of connectionist parsing, one more similar to classical deterministic parsers, as described in Marcus [26], for example.

6 Conclusion

This article has described the use of Simple Synchrony Networks (SSNs) for learning to parse samples of English sentences drawn from a corpus of naturally occurring text. We have described an input-output representation which enables the SSN to incrementally output the parse tree of a sentence. This representation is important in demonstrating how a connectionist architecture can manipulate hierarchical and recursive output representations. We have also introduced an important mechanism for improving the $O(n^2)$ speed of the basic SSN architecture to linear time. This mechanism, based on the concept of a short-term memory (STM), enables the SSN to retain only the necessary constituents for processing. In the experiments, we have demonstrated that a number of SSN architectures provide reliable generalization performance in this domain.

The theoretical and experimental results of this article go beyond language learning. It is apparent that the SSN architectures, modeled on the Simple Recurrent Network, are not specifically adapted to natural language. Thus, their ability to learn about and manipulate structured information is a general one. Although the specific input-output representation used in these experiments is carefully tailored to the target domain, its underlying principles of incremental output and recursively defined structures may be applied more widely. Also, the STM queue, although again defined based on cognitive limitations on language processing, may also be used in further domains where appropriate.

Acknowledgments

The authors would like to thank Fernand Gobet, the reviewers, and editors for their helpful comments on this article. This work was carried out while Peter Lane was at the Department of Computer Science, University of Exeter, funded by the Engineering and Physical Sciences Research Council, UK. We acknowledge the roles of the Economic and Social Research Council (UK) as a sponsor and the University of Sussex as a grantholder in providing the SUSANNE corpus used in the experiments reported in this article.

References

[1] A.D. Baddeley, *Working Memory.* New York: Oxford Univ. Press, 1986.
[2] E. Charniak, *Statistical Language Learning.* Cambridge, Mass.: MIT Press, 1993.
[3] E. Charniak, "Statistical Techniques for Natural Language Parsing," *AI Magazine,* vol. 18, pp. 33–43, 1997.
[4] N. Chomsky, "On Certain Formal Properties of Grammars," *Information and Control,* vol. 2, pp. 137–167, 1959.
[5] M. Collins, "Head-Driven Statistical Models of Natural Language Parsing," PhD thesis, Univ. of Pennsylvania, Philadelphia, 1999.
[6] J.L. Elman, "Finding Structure in Time," *Cognitive Science,* vol. 14, pp. 179–211, 1990.
[7] J.L. Elman, "Distributed Representations, Simple Recurrent Networks, and Grammatical Structure," *Machine Learning,* vol. 7, pp. 195–225, 1991
[8] J.L. Elman, "Learning and Development in Neural Networks: The Importance of Starting Small," *Cognition,* vol. 48, pp. 71–99, 1993.
[9] J.A. Fodor and Z.W. Pylyshyn, "Connectionism and Cognitive Architecture: A Critical Analysis," *Cognition,* vol. 28, pp. 3–71, 1988.
[10] P. Frasconi, M. Gori, and A. Sperduti, "On the Efficient Classification of Data Structures by Neural Networks," *Proc. Int'l Joint Conf. Artificial Intelligence,* 1997.
[11] P. Frasconi, M. Gori, and A. Sperduti, "A General Framework for Adaptive Processing of Data Structures," *IEEE Trans. Neural Networks,* vol. 9, pp. 768–786, 1998.
[12] *The Computational Analysis of English: A Corpus-Based Approach.* R. Garside, G.Leech, and G. Sampson, eds., Longman Group United Kingdom Limited, 1987.
[13] R.F. Hadley and M.B. Hayward, "Strong Semantic Systematicity from Hebbian Connectionist Learning," *Minds and Machines,* vol. 7, pp. 1–37, 1997.
[14] J.B. Henderson, "Description Based Parsing in a Connectionist Network," PhD thesis, Univ. of Pennsylvania, 1994.
[15] J.B. Henderson, "A Connectionist Architecture with Inherent Systematicity," *Proc. 18th Conf. Cognitive Science Soc.,* pp. 574–579, 1996.
[16] J.B. Henderson, "A Neural Network Parser that Handles Sparse Data," *Proc. Sixth Int'l Workshop Parsing Technologies,* pp. 123–134, 2000.
[17] J.B. Henderson and P.C.R. Lane, "A Connectionist Architecture for Learning to Parse," *Proc. 17th Int'l Conf. Computational Linguistics and the 36th Ann. Meeting of the Assoc. for Computational Linguistics (COLING-ACL'98),* pp. 531–537, 1998.
[18] E.K.S. Ho and L.W. Chan, "How to Design a Connectionist Holistic Parser," *Neural Computation,* vol. 11, no. 8, pp. 1995–2016, 1999.
[19] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation,* vol. 9, no. 8, pp. 1735–1780, 1997.
[20] M. Johnson, "PCFG Models of Linguistic Tree Representations," *Computational Linguistics,* vol. 24, pp. 613–632, 1998.
[21] P.C.R. Lane, "Simple Synchrony Networks: Learning Generalizations across Syntactic Constituents," *Proc. 13th European Conf. Artificial Intelligence,* H. Prade, ed., pp. 469–470, 1998.

[22] P.C.R. Lane, "Simple Synchrony Networks: A New Connectionist Architecture Applied to Natural Language Parsing," PhD thesis, Dept. Computer Science, Univ. of Exeter, England, 2000.

[23] P.C.R. Lane and J.B. Henderson, "Simple Synchrony Networks: Learning to Parse Natural Language with Temporal Synchrony Variable Binding," *Proc. Eighth Int'l Conf. Artificial Neural Networks,* pp. 615–620, 1998.

[24] S. Lawrence, S. Fong, and C.L. Giles, "Natural Language Grammatical Inference: A Comparison of Recurrent Neural Networks and Machine Learning Methods," *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing,* S. Wermter, E. Riloff, and G. Scheler, eds., 1996.

[25] S. Lawrence, C.L. Giles, and S. Fong, "Natural Language Grammatical Inference with Recurrent Neural Networks," *IEEE Trans. Knowledge and Data Eng.*, vol. 12, no. 1, Jan./Feb. 2000.

[26] M. Marcus, *A Theory of Syntactic Recognition for Natural Language.* Cambridge Mass.: MIT Press, 1980.

[27] I. Melcuk, *Dependency Syntax: Theory and Practice.* SUNY Press, 1988.

[28] G.A. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," *Psychological Rev.,* vol. 63, pp. 81–97, 1956.

[29] *Proc. Eighth Int'l Conf. Artificial Neural Networks,* L. Niklasson, M. Boden, and T. Ziemke, eds., 1998.

[30] R.G. Reilly, "Enriched Lexical Representations, Large Corpora, and the Performance of SRNs," *Proc. Eighth Int'l Conf. Artifical Neural Networks,* pp. 405–410, 1998.

[31] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning Internal Representations by Error Propagation," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition,* D.E. Rumelhart, J.L. McClelland, and the PDP Research Group, eds., vol. 1, 1986.

[32] G. Sampson, *English for the Computer.* Oxford, United Kingdom: Oxford Univ. Press, 1995.

[33] L. Shastri and V. Ajjanagadde, "From Simple Associations to Systematic Reasoning: A Connectionist Representation of Rules, Variables, and Dynamic Bindings using Temporal Synchrony," *Behavioral and Brain Sciences,* vol. 16, pp. 417–494, 1993.

[34] A. Sperduti, "Stability Properties of Labeling Recursive Auto-Associative Memory," *IEEE Trans. Neural Networks,* vol. 6, pp. 1452–1460, 1995.

[35] A. Sperduti and T. Starita, "Supervised Neural Networks for Classification of Structures," *IEEE Trans. Neural Networks,* vol. 8, pp. 714–735, 1997.

[36] C. von der Malsburg, "The Correlation Theory of Brain Function," Technical Report 81-2, Max-Planck-Inst. for Biophysical Chemistry, Gottingen, 1981.

Peter C.R. Lane received the BA degree in mathematics and computation from the University of Oxford in 1991, the MSc degree in computer science from the University of Exeter in 1995, and the PhD degree in computer science from the University of Exeter in 2000. Since October 1998, he has worked at the Centre for Research in Development, Instruction, and Training at the University of Nottingham. His research interests are centered on topics in machine learning: computational linguistics, connectionist networks and, recently, the cognitive modeling of the learning process using a form of instance-based learning.

James B. Henderson received the BSc degree in computer science from the Massachusetts Institute of Technology in 1987, the MSE degree in computational linguistics from the University of Pennsylvania and, in 1994, the PhD degree. He is a lecturer in the Department of Computer Science at the University of Exeter, United Kingdom, where he works on neural networks and computational linguistics. For the 2000/2001 academic year, he is on leave at the Centre Universitaire d'Informatique at the University of Geneva, Switzerland.