A NEURAL NETWORK PARSER THAT HANDLES SPARSE DATA*

James Henderson
University of Exeter
School of Engineering and Computer Science
Exeter EX4 4PT, UK
J.B.Henderson@exeter.ac.uk

Abstract

Previous work has demonstrated the viability of a particular neural network architecture, Simple Synchrony Networks, for syntactic parsing [6]. Here we present additional results on the performance of this type of parser, including direct comparisons on the same dataset with a standard statistical parsing method, Probabilistic Context Free Grammars. We focus these experiments on demonstrating one of the main advantages of the SSN parser over the PCFG, handling sparse data. We use smaller datasets than are typically used with statistical methods, resulting in the PCFG finding parses for under half of the test sentences, while the SSN finds parses for all sentences. Even on the PCFG's parsed half, the SSN performs better than the PCFG, as measure by recall and precision on both constituents and a dependency-like measure.

1 Introduction

In many domains neural networks are an effective alternative to statistical methods. This has not been the case for syntactic parsing, but recent work has identified a viable neural network architecture for this problem (SSN) [6], [7]. This alternative parsing technology is potentially of interest because neural networks have different strengths from statistical methods, and thus may be more applicable to some tasks. Like statistical methods, neural networks are robust against noise in the input and errors in the data. But unlike statistical methods, neural networks are particularly good at handling sparse data. In order to compensate for the necessarily limited amount of data available, statistical methods must make strong independence assumptions. These assumptions lead to undesirable biases in the model generated, and may still not guarantee coverage of less frequent cases. Neural networks also require independence assumptions in order to define their input-output format, but these assumptions can be much weaker. Because neural networks learn their own internal representations, neural networks can decide automatically what features to count and how reliable they are for predicting the output.

In this paper we empirically investigate the ability of SSNs to handle sparse data, relative to the statistical method Probabilistic Context Free Grammars (PCFGs) and a statistical version of the SSN parser. To test this ability we use a small dataset relative to those typically used with statistical methods. We train all the models on the same dataset and compare their results, both in terms of coverage and performance on the covered portion. The statistical version of the SSN parser has good coverage, but its performance is worse than both the other two models. The PCFG covers under half of the test sentences. The SSN produces a parse for all of the sentences, while still achieving

[0]Published in *Proceedings of the 6th International Workshop on Parsing Technologies*, Trento, Italy, 2000.

better performance than that of the PCFG on the PCFG's parsed sentences, as measure by recall and precision on both constituents and a dependency-like measure.

2 Simple Synchrony Networks

The neural network architecture used in this paper has previously been shown to be a viable parsing technology based on both initial empirical results [6] and the linguistic characteristics of the basic computational model [5]. Their appropriateness for application to syntactic parsing is a result of their ability to learn generalisations over structural constituents. This generalisation ability is a result of using a neural network method for representing entities, called Temporal Synchrony Variable Binding (TSVB) [12]. In Simple Synchrony Networks (SSNs) this method is used to extend a standard neural network architecture for processing sequences, Simple Recurrent Networks (SRNs) [3]. SRNs can learn generalisations over positions in an input sequence (and thus can handle unbounded input sequences), which has made them of interest in natural language processing. However, their output at any given time in the input sequence is an unstructured vector, thus making them inappropriate for recovering the syntactic structure of an input sentence. By using TSVB to represent the constituents in a syntactic structure, SRNs can be extended to also learn generalisations over structural constituents. The linguistic relevance of this class of generalisations is what accounts for the fact that SSNs generalise from training set to testing set in an appropriate way, as demonstrated in section 4. In this section we briefly outline the SSN architecture.

2.1 Representing Constituents

The problem of representing constituents in a neural network is an instance of a broader problem, representing entities in general. The difficulty is that each entity can have multiple features and the network needs to represent which features go with which entity. One proposal is to use pulsing units and represent the binding between the features of each entity using the synchrony of the units' activation pulses. This has been proposed on both biological grounds [13] and computational grounds [12]. Following the computational work we will call this method Temporal Synchrony Variable Binding. We use TSVB's pulsing units to represent information about syntactic constituents.

TSVB can be applied to a learning task by combining it with a standard neural network architecture. Because we need the network to process sequences as well as constituents, we use an architecture that combines TSVB with Simple Recurrent Networks. In addition to a layer of input units and a layer of output units, SRNs have a layer of memory units which store the internal state of the network from the previous position in the sequence. This allows SRNs to learn their own internal representation of the previous input history. To extend SRNs with TSVB, pulsing units need to be added. These units represent inputs, outputs, and internal state about individual constituents. In order to retain the SRN's ability to represent information about the situation as a whole, the architecture also has normal nonpulsing units. These units also provide a means for information about one constituent to be passed to other constituents.

Pulsing units can be added to SRNs in a variety of ways. One proposed class of such networks is called Simple Synchrony Networks (SSNs) [7]. The architectural restriction which defines SSNs is appropriate for the parser used here because information about only one constituent is input at any one time, namely the constituent headed by the current input word. This input simply specifies the

current word. To accommodate SSN's restrictions we simply need to provide the same information through nonpulsing input units as we provide to this one constituent through the pulsing input units [6]. In this way each constituent gets information about its head word from the pulsing input units and gets information about every other word from the nonpulsing input units.

In this paper we use the simplest form of SSN, called type A in [7]. This architecture only has a single internal state layer and a single memory layer, both consisting of pulsing units. We use 100 units in each of these layers, which is a moderately large size, resulting in a ability to approximate a wide variety of functions.

2.2 Learning Generalisations over Constituents

The most important feature of any learning architecture is how it generalises from training data to testing data. SRNs are popular for sequence processing because they inherently generalise over sequence positions. Because inputs are fed to an SRN one at a time, and the same trained parameters (the link weights) apply at every time, information learned at one sequence position will inherently be generalised to other sequence positions. This generalisation ability manifests itself in the fact that SRNs can handle arbitrarily long sequences. The same argument applies to TSVB and constituents. Using synchronous pulses to represent constituents amounts to cycling through the set of constituents and processing them one at a time. Because the same trained parameters are applied at every time, information learned for one constituent will inherently be generalised to other constituents. This generalisation ability manifests itself in the fact that these networks can handle arbitrarily many constituents.

3 A SSN Parser

The Simple Synchrony Network architecture gives us the basic generalisation abilities that we need for syntactic parsing, but the specific way that a SSN parser generalises from training set to testing set also depends on the specific input-output format that we ask it to learn. The main challenge in defining an input-output format is that a syntactic structure consists of a set of relationships between constituents, whereas SSNs only provide us with outputs about individual constituents. More precisely, there are $O(n^2)$ possible relationships between constituents and only $O(n)$ outputs at any given time. The solution is simply to output the syntactic structure incrementally. By making use of the $O(n)$ positions in the input sentence[1] to produce output, the network can produce the necessary $O(n^2)$ outputs over the course of the parse. In this section we will specify the nature of this incremental output in more detail, and give specifics about the format of the input to the network. In each case, the importance of these choices is how they effect the way that the network will generalise.

3.1 The Model of Constituency

Before discussing the specific input-output pattern, it is necessary to specify what exactly we mean by "constituent". This is a crucial decision, since the constituent is the unit over which generalisations are learned. One answer would be to simply use the definition of constituency that is implicit in whatever corpus you have available. It would be possible for us to do this, but at the cost of a more

[1] Here we are assuming that the number of constituents is linear in the number of words. This is uncontroversial, being implied by any lexicalised or dependency-based grammar.

complicated output format. Also, the import of constituents as the unit of generalisations may not be the one intended by the corpus writer. Instead we choose to take a lexicalised approach. Each constituent is associated with a word in the input, which is intended to be the head of that constituent. The constituent structure can then be thought of as a set of relationships between each constituent's head word and the other words and constituents in the structure. This perspective is closely related to Dependency Grammar [8]. It is also closely related to Lexicalized Tree Adjoining Grammar [11] and the head-driven statistical parsing model by Collins [2], but in these cases our constituent needs to be mapped to the entire projection of a lexical head.

Unfortunately the corpus we will use in the below experiments does not label heads for constituents. As an approximation, because we know we are using a head-initial language, we will use the first word which is a direct child of a constituent as the constituent's head. If a constituent in the original corpus does not have any words as direct children, then that constituent is collapsed into one of its child constituents, as discussed further in section 4.1. Thus the corpus used in the below experiments is slightly flatter than most corpora. In particular it often does not distinguish between an S and its VP, the two constituents having been collapsed.

Using a lexicalised definition of constituency not only makes the unit of generalisation more linguistically appropriate, it has the added advantage that it provides a fixed mapping between the constituents in the network's output and the constituents in the corpus. Two constituents are the same if they have the same word as their head. Using this mapping it is easy to define a fixed input-output pattern for the network to learn during training, which is necessary for supervised learning algorithms such as the one used here. However, we should emphasise that it would be possible to define a different input-output format for an SSN parser which could use any corpus's definition of constituency.

3.2 The Independence Assumptions

As mentioned above, the independence assumptions made by a neural network model are embodied in the definition of its input-output format. If there is no way for information to flow from a given input to a given output, then the model is assuming that that output is independent of that input. For the SSN parser used here these assumptions are due to the incremental nature of the network's output.

As discussed above, the SSN parser solves the problem of outputting relationships between constituents by outputting them incrementally. In particular, this SSN parser outputs structural relationships maximally incrementally. Words are input to the parser one at a time, and with each word a new constituent is introduced to act as the constituent which the word heads, if needed.[2] A structural relationship is output as soon as both things in the relationship have been introduced. This incremental output is illustrated in figure 1. When the parse is complete the accumulation of all the outputs fully specifies the parse, as illustrated in figure 1.

The syntactic structure of a sentence can be fully specified by specifying the set of parent-child relationships in it. These come in two types, one where the parent is a constituent but the child is a word, and one where both the parent and the child are constituent. When the child is a word, then because we assume that the first word child of a constituent is its head, the parent constituent

[2] In the actual experiments below we use part-of-speech tags as inputs, not the actual words. We are using "words" here for expository convenience and clarity.

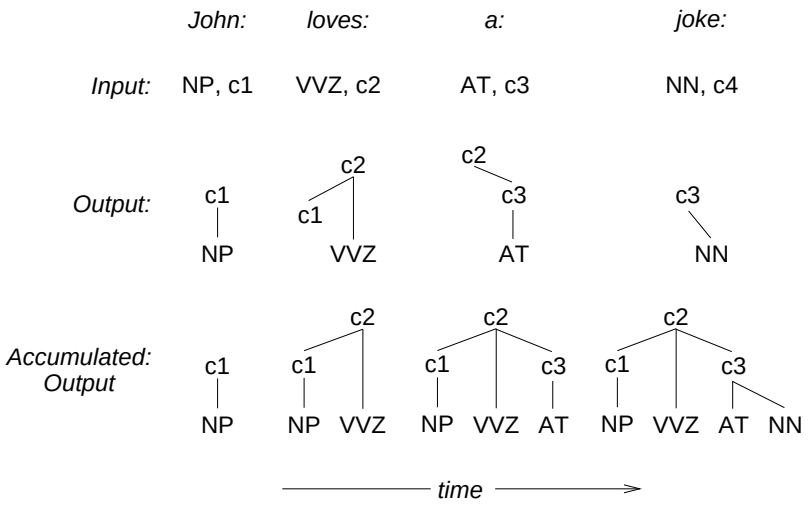

Figure 1: An example of the SSN parser's output.

must have been introduced before or at the same time as the word is input. Thus as soon as a word is input, the network can output which constituent is the parent of that word. This is done with a single output unit, called the *parent* output, which pulses in synchrony with the parent of the current input word. For example when 'John' is input in figure 1 the new constituent, *c1*, is specified as the parent, while when 'joke' is input the previous constituent, *c3*, is specified as the parent. When both the parent and child are constituents, then the relationships is output as soon as the second of the two constituents is input. If the second constituent is the child, then this output is specified using the *grandparent* output unit, which pulses in synchrony with the parent of the constituent which is headed by the current input word. For example when 'a' is input in figure 1 the previous constituent, *c2*, is specified as the grandparent. If the second constituent is the parent, then this output is specified using the *sibling* output unit, which pulses in synchrony with the child of the constituent which is headed by the current input word. For example when 'loves' is input in figure 1 the previous constituent, *c1*, is specified as a sibling. Given the previous assumptions, these three output units are all that is necessary to specify the syntactic structure of the sentence.

So far we have only discussed the output pattern which we would like the network to produce (the target output), but a network will actually output real values, not zeros and ones. To interpret this output we look at all the possible parents for a given child, and pick the one whose output activation is the highest. First we incrementally choose the parents for each word, based on the *parent* output unit's activations. This not only determines all parent-child relationships involving words, it also determines which of the introduced constituents have head words and thus are included in the output structure. For example in figure 1 constituent *c4* is not included in the output structure. Only these headed constituents candidates to be in relationships with other constituents or later words. Second we choose the best parent for these headed constituents, based on both the *grandparent* output unit's activations at the time of the constituent's head (for leftward attachments) and the *sibling* output unit's activations in synchrony with the constituent (for rightward attachments). This determines all the parent-child relationships between two constituents.[3] While the resulting set of parent-child

[3] There is one distinguished constituent, introduced with a sentence-initialising input symbol, which does not require a parent. This acts as the root of the parse tree, in the same way as the start symbol of CFGs.

relationships may not form a tree, this representation of the structure is sufficient to determine which words are included in each constituent, and thus to calculate recall and precision on constituents.

This maximally incremental output format implies some fairly strong independence assumptions. This is particularly true for the parents of words. No output after a word can influence which constituent is interpreted as its parent, and thereby this decision is assumed to be independent of later input words. For the parents of constituents there is a slight complication in that a later *sibling* output can override a *grandparent* output that is produced at the time of the constituent's head word. For example, in the sentence "John knows Mary left", the network may produce a fairly high *grandparent* output value for attaching 'Mary' to 'knows', but when 'left' is input an even higher *sibling* output value can override this and result in 'Mary' being attached to 'left'. However it is not possible for later outputs to affect a decision which does not involve a later word's constituent. For example, in the sentence "John saw the policeman with Mary", the decision of whether 'with' should be attached to 'saw' or 'the policeman' is not effected by any output at the time when 'Mary' is introduced, and thus the unsuitability of 'Mary' as an instrument of the seeing cannot be used. Thereby decisions between two possible parents for a constituent are assumed to be independent of any input words after the last head word of the three constituents involved.

Finally, in addition to outputs about structural relationships, the parser also outputs the labels of the constituents. This output is produced at the time when the constituent's head word is input, using a bank of output units, one for each label. Thus the decision of what label to give a constituent is assumed to be independent of all words after the constituent's head word. This assumption is mostly for convenience, since any output about an individual constituent could be delayed until the end of the sentence.[4]

3.3 The Soft Biases

Because all the words are input to the representation of each constituent, in theory any earlier input can effect a later output, and thus they are not being assumed to be independent. However, in practice there are ways to make the network pay more attention to some inputs than to others. In particular, a recurrent network such as an SSN will learn more about the dependencies between an input and an output if they are close together in time. The immediate effect of this is a form of recency bias; the most recent input words will effect an output more than earlier input words. To make use of this we provide a *new constituent* input pattern, which is correlated with being an output for a short time afterwards. We also provide a *last parent* input unit, which is the disambiguated *parent* output from the previous input word. This is also correlated with being an output for a short time afterwards. Finally, we bias the network to pay particular attention to the head word for each constituent by providing the head word as an input at every time during the life of a constituent. Thus an output at a given input word for a given constituent pays particular attention to the input word and the head of the constituent, with previous input words providing an influence proportional to their recency. This input format has been devised in part on the basis of the author's linguistic knowledge and in part on the basis of experimental results.[5]

[4] Because in the experiments below we are actually inputting part-of-speech tags and there are a fairly small number of possible labels, labelling of constituents is actually not difficult and thus this independence assumption is not a problem.

[5] All development decisions such as these have been made on the basis of the cross validation set used below. The testing set was not used until the final results reported in section 4.3 were produced. The one exception was a set of results collected before training had completed for the submitted version of this paper.

4 Generalising from Sparse Data

To test the ability of Simple Synchrony Networks to handle sparse data we train the SSN parser described in the previous section on a relatively small set of sentences and then test how well it generalises to a set of previously unseen sentences. Because the training set is small, the testing sentences will contain many constituents and constituent contexts which the network has never seen before. Thus the network cannot simply choose the constituents which it has seen the most times in the training set, because in most cases it will have never seen any of the constituents which it needs to choose between. To handle this sparseness of training data the network must learn which of the inputs about a constituent are important, as well as how they correlate with the outputs. The advantage of SSNs is that they automatically learn the relative importance of different inputs as part of their training process.

To provide a comparison with the SSN parser we also apply a standard statistical method to the same data sets. We estimate the probabilities for two Probabilistic Context Free Grammars, one using just the network's training sentences and another using both the training sentences and the cross validation sentences. These PCFGs are then both tested on the network's testing sentences. PCFGs deal with the problem of sparse data by ignoring everything about a constituent other than its label. The strength of this independence assumption depends on how many constituent labels the corpus has, and thus how much information the labels convey. Because we are dealing with small training sets, we only use a small number of labels. Even so, the PCFGs have only seen enough CFG rules in the training sets to produce parses for about half of the test sentences. The problem with such statistical approaches is that information is either counted (i.e. increasing the number of labels) or ignored (i.e. decreasing the number of labels), and there is no middle ground.[6]

In addition to the PCFGs, we train a statistical model based on the output format of the SSN parser. This test is to control for the possibility that it is the linguistic assumptions discussed in the previous section which are responsible for the SSN parser's performance, and not the SSN architecture. Rather than estimating the probabilities for CFG rules like in PCFGs, this statistical model estimates the probabilities of the same structural relationships used in the output of the SSN parser (parent, grandparent, and sibling, plus label-head relationships). Thus we will call this model the Probabilistic Structural Relationships (PSR) model. The PSR model also makes all the independence assumptions made by the SSN parser, but in order to deal with the sparse data it must also make additional independence assumptions. Every structural relationship is dependent on the word involved in the relationship and the head word of the constituent involved in the relationship (and whether they are the same), but they are independent of all other words. This assumption imposes a hard bias that parallels the main soft biases provided by the SSN parser's input format. This independence assumption is strong enough to provide us with sufficient statistics given our training data, but still captures to the extent possible the relevant information for estimating the structural relationship probabilities. The PSR model is closely related to dependency-based statistical models, such as that in [2].

[6] It should be noted that we are using the term "statistical method" here in a rather narrow sense, intending to reflect common practice in parsing technology. Indeed, neural networks themselves are a statistical method in the broader sense. We do not intend to imply that there are no other methods for addressing the problem of sparse data. For example, Maximum Entropy is a framework for deciding how strongly to believe different sources of information and has been applied to parsing [9], although with a much larger training set than that used here.

4.1 A Small Corpus

Work on statistical parsing typically uses a very large corpora of preparsed sentences (for example more than a million words in [1], [2], and [9]). Such corpora are very expensive to produce, and are currently only available for English. In addition, using a very large training corpus helps hide inadequacies in the parser's ability to generalise, because the larger the training corpus the better results can be obtained by simply memorising the common cases.[7] Here we use a training set of only 26,480 words, plus a cross validation set of 4365 words (30,845 words total). By using a small training set we are placing greater emphasis on the ability of the parser to generalise to novel cases in a linguistically appropriate way, and to do so robustly. In other words, we are testing the parser on its ability to deal with sparse data.

We use the Susanne[8] corpus as our source of preparsed sentences. The Susanne corpus consists of a subset of the Brown corpus, preparsed according to the Susanne classification scheme described in [10], and we make use of the "press reportage" subset of this. These parses have been converted to a format appropriate for this investigation, as described in the rest of this section.[9]

As is commonly done for PCFGs, we do not use words as the input to the parser, but instead use part-of-speech tags. The tags in the Susanne scheme are a detailed extension of the tags used in the Lancaster-Leeds Treebank (see [4]), but we use the simpler Lancaster-Leeds scheme. Each tag is a two or three letter sequence, for example 'John' would be encoded 'NP', the articles 'a' and 'the' are encoded 'AT', and verbs such as 'is' encoded 'VBZ'. There are 105 tags in total.

The parse structure in the Susanne scheme is also more complicated than is needed for our purposes. Firstly, the meta-sentence level structure has been discarded, leaving only the structures of individual sentences. Secondly, the 'ghost' markers have been removed. These elements are used to represent long distance dependencies, but they are not needed here because they do not effect the boundaries of the constituents. Third, as was discussed above, we simplify the nonterminal labels so as to help the PCFG deal with the small training set. We only use the first letter of each label, resulting in 15 nonterminal labels (including a new start symbol). Finally, as was discussed in section 3.1, some constituents need to be collapsed with one of their child constituents so that every constituent has at least one terminal child. There are very few constructions in the Susanne corpus that violate this constraint, but one of them is very common, namely the S-VP division. The head word of the S (the verb) is within the VP, and thus the S often occurs without any terminals as immediate children. In these cases, we collapse the S and VP into a single constituent, giving it the label S. The same is done for other such constructions. As discussed above, this change is not linguistically unmotivated.

The total set of converted sentences was divided into three disjoint subsets, one for training, one for cross validation, and one for testing. The division was done randomly, with the objective of producing cross validation and testing sets which are each about an eighth of the total set. No restrictions were placed on sentence length in any set. The training set has 26480 words, 15411 constituents, and 1079 sentences, the cross validation set has 4365 words, 2523 constituents, and 186 sentences, and the testing set has 4304 words, 2500 constituents, and 181 sentences.

[7] Given this fact, it is unfortunate that much work on statistical parsing does not even report the number of words in their training set.

[8] We acknowledge the roles of the Economic and Social Research Council (UK) as sponsor and the University of Sussex as grant holder in providing the Susanne corpus used in the experiments described in this paper.

[9] We would like to thank Peter Lane for performing most of this conversion.

4.2 Training the Models

The SSN parser was trained using standard training techniques extended for pulsing units [7]. Neural network training is an iterative process, in which the network is run on training examples and then modified so as to make less error the next time it sees those examples. This process can be continued until no more changes are made, but to avoid over-fitting it is better to check the performance of the network on a cross validation set and stop training when this error reaches a minimum. This is why we have split the corpus into three datasets, one for training, one for cross validation, and one for testing. This technique also allows multiple versions of the network to be trained and then evaluated using the cross validation set, without ever using the testing set until a single network has been chosen. This is the technique which we have used in developing the specific network design reported in this paper. The resulting network trained for 325 passes through the training set before reaching a maximum of the average between its recall and precision on constituents in the cross validation set.

Estimating the parameters of a PCFG is straightforward. All the sequences of child labels that occur in the corpus for each parent label need to be extracted, counted, and normalised in accordance with the conditional probabilities required by the model. Because this process does not require a cross validation set, we create two PCFGs, one using only the network's training set and the other using both the training set and the cross validation set. The first provides a more direct comparison with the SSN parser's ability to deal with small training sets, but the second provides a better indication of how the two methods compare in practice.

Estimating the PSR model is also straightforward. All the head tag bigrams associated with each structural relationship (or label-head tag bigrams) are extracted, counted, and normalised in accordance with the probability model.[10] As with the second PCFG, we use the network's training set plus its cross validation set to estimate the probabilities.

In addition to embodying the same linguistic assumptions as the SSN parser, the PSR model has the advantage that it has a finite space of possible parameters (namely one probability per tag bigram for each relationship). Because a PCFG places no bound on the number of children that a parent constituent can have, a PCFG has an infinite space of possible parameters (namely one probability for each of the infinite number of possible rules). This makes it difficult to apply smoothing to a PCFG, to avoid the problem of assigning zero probability to rules that did not occur in the training set. Thus we have not applied smoothing to the PCFGs, contributing to the bad test set coverage discussed in the next section. However the PSR's finite space of parameters makes it simple to apply smoothing to the PSR model. Thus we apply a smoothing method to estimate the parameters for a second PSR model; before normalising we add half to all the counts so that none of them are zero.

4.3 Testing Results

Once all development and training had been completed, the SSN parser, the two PCFGs, and the two PSRs were tested on the data in the testing set. For the PCFGs and the PSRs the most probable parse according to the model was taken as the output of the parser.[11] The results of this testing are shown in table 1, where PCFG 1 is the PCFG that was produced using only the training set, PCFG 2

[10] Briefly, in the probability model each structural relationship probability is conditional on the constituents' heads being correct, and the chain rule plus the independence assumptions from section 4 are used to combine these probabilities into the probability of the entire parse.

[11] We would like to thank Jean-Cedric Chappelier and the LIA-DI at EPFL, Lausanne, Switzerland for providing the tools used to train and test the PCFG.

	Sentences		Constituents		Parent-child	
	Parsed	Correct	Recall	Precision	Recall	Precision
SSN	100%	14.4%	65.1%	65.0%	83.0%	82.9%
PCFG 1	45.3%	3.3%	24.9%	54.0%	32.2%	72.4%
PCFG 2	50.8%	3.3%	29.2%	53.7%	38.2%	73.3%
PSR	90.6%	2.8%	37.0%	40.3%	63.1%	65.1%
PSR Sm	100%	2.8%	35.9%	36.8%	58.8%	59.4%

Table 1: Testing results.

		Sentences	Constituents		Parent-child	
		Correct	Recall	Precision	Recall	Precision
Parsed by PCFG 1	SSN	19.5%	66.6%	67.6%	83.7%	84.0%
	PCFG 1	7.3%	58.1%	54.0%	74.3%	72.4%
Parsed by PCFG 2	SSN	17.4%	65.4%	66.4%	83.4%	83.8%
	PCFG 2	6.5%	57.5%	53.7%	75.2%	73.3%

Table 2: Testing results on the sentences parsed by PCFG 1 and PCFG 2, respectively.

is the PCFG produced using both the training set and the cross validation set, PSR is the unsmoothed PSR, and PSR Sm is the smoothed PSR.

The first thing to notice about the testing results is that both PCFGs only found parses for about half of the sentences. For the unparsed sentences the PCFGs had not found enough rules in their training sets to construct a tree that spans the entire sentence, and thus there is no straightforward way to choose the most probable parse.[12] In contrast the SSN parser is able to make a guess for every sentence. This is a result of the definition of the SSN parser, as with the smoothed PSR, but unlike both PSR models the SSN parser produces good guesses. The first evidence of the quality of the SSN parser's guesses is that they are exactly correct more that 4 times as often as for any of the PCFGs or PSRs.

The remaining four columns in table 1 give the performance of each parser in terms of recall (percentage of desired which are output) and precision (percentage of output which are desired) on both constituents and parent-child relationships. An output constituent is the same as a desired constituent if they contain the same words and have the same label. This measure is a common one for comparing parsers. Parent-child relationships are the result of interpreting the parse as a form of dependency structure. Each parent-child relationship in the parse is interpreted as a dependency from the head word of the child to the head word of the parent. Two such relationships are the same if their words are the same. This measure is more closely related to the output of the SSN parser and the PSR model, and may be more appropriate for some applications.

Given that both PCFGs produce no parse for about half of the sentences, it is no surprise that the SSN parser achieves about twice the recall of both PCFGs on both constituents and parent-child relationships (although it is interesting that the SSN actually performs more than twice as well). Thus we also compute these performance figures for the subset of sentences which are parsed by each PCFG,

[12] It would be possible to use methods for choosing partial parses, and thus improve the recall results in table 1 (at the cost of precision). Instead we choose to report results on the parsed subset, since this is sufficient to make our point.

as discussed below. However restricting attention to the parsed subset will not change the PCFGs' precision figures. Just as the recall figures are particularly low, the precision figures are improved due to the fact that the PCFGs are not outputting anything for those sentences which are particularly hard for them (i.e. the sentences they cannot parse). Even so, the SSN does about 10% better than either PCFG on both constituent precision and parent-child precision.

Table 2 shows the performance of the PCFGs and SSN on the subset of test sentences parsed by each PCFG. Note that these figures are biased in favour of the PCFGs, since we are excluding only those sentences which are difficult for each PCFG, as determined by the results on the testing set itself. Even so, the SSN outperforms each PCFG under every measure. In fact, under every measure the SSN's performance on the entire testing set is better than the PCFGs' performance on the subset of sentences which they parse.

Given the large difference between the linguistic assumptions embodied in the PCFGs and the SSN parser, we need to address the possibility that the better performance of the SSN parser on this corpus is due to its linguistic assumptions and not due to the SSN architecture. The poor performance of both PSR models clearly demonstrates this. The PSR model was designed to follow the linguistic assumptions embodied in the SSN parser as closely as possible, only imposing additional independence assumptions to the extent that they were required to get sufficient counts for estimating the model's probabilities. Nonetheless, both PSR models do much worse than the PCFGs, even on the parent-child relationships, which are directly related to the parameters of the PSR model. The only exception is comparing against the recall results of the PCFG models including their unparsed sentences, but the recall results of the PCFG models on the parsed subsets indicates that this is simple an artifact of the low coverage of the PCFGs. Thus the better performance of the SSN parser cannot be due to its linguistic assumptions alone. It must be due to the SSN architecture's ability to handle sparse data without the need to impose strong independence assumptions.

One final thing to notice about these results is that there is not a big difference between the results of the SSN parser on the full testing set and the results on the subsets which are parsed by each PCFG. There is a small improvement, reflecting the fact that the PCFGs fail on the more difficult sentences. However the lack of any big improvement is a further demonstration that the SSN is not simply returning parses for every sentence because it is defined in such a way that it must do so. The SSN is making good guesses, even on the difficult sentences. This demonstrates the robustness of the SSN parser in the face of sparse training data and the resulting novelty of testing cases.

5 Conclusion

The good performance of the Simple Synchrony Network parser despite being trained on a small training set demonstrates that SSN parsers are good at handling sparse data. In comparison with Probabilistic Context Free Grammars trained on the same data, the SSN parser not only returns parses for twice as many sentences, its performance on the full testing set is even better than the performance of the PCFGs on the subset of sentences which they parse.

By demonstrating SSNs' ability to handle sparse data we have in fact shown that SSNs generalise from training data to testing data in a linguistically appropriate way. The poor performance of the PSR model shows that this is not simply due to clever linguistic assumptions embodied in the particular parser used. This generalisation performance is due to SSNs' ability to generalise across constituents as well as across sequence positions, plus the ability of neural networks in general to

learn what input features are important as well as what they imply about the output. The resulting robustness makes SSNs appropriate for a wide variety of parsing applications, particularly when a small amount of data is available or there is a large amount of variability in the input.

References

[1] Eugene Charniak. Statistical techniques for natural language parsing. *AI Magazine*, 1997.

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

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

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

[5] James Henderson. *Description Based Parsing in a Connectionist Network*. PhD thesis, University of Pennsylvania, Philadelphia, PA, 1994. Technical Report MS-CIS-94-46.

[6] James Henderson and Peter Lane. A connectionist architecture for learning to parse. In *Proceedings of COLING-ACL*, pages 531–537, Montreal, Quebec, Canada, 1998.

[7] Peter Lane and James Henderson. Simple synchrony networks: Learning to parse natural language with temporal synchrony variable binding. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 615–620, Skovde, Sweden, 1998.

[8] I. Melčuk. *Dependency Syntax: Theory and Practice*. SUNY Press, 1988.

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

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

[11] Yves Schabes. *Mathematical and Computational Aspects of Lexicalized Grammars*. PhD thesis, University of Pennsylvania, Philadelphia, PA, 1990.

[12] Lokendra Shastri and Venkat Ajjanagadde. From simple associations to systematic reasoning: A connectionist representation of rules, variables, and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences*, 16:417–451, 1993.

[13] C. von der Malsburg. The correlation theory of brain function. Technical Report 81-2, Max-Planck-Institute for Biophysical Chemistry, Gottingen, 1981.