To appear in *Neural Processing Letters*, Vol. 21, No 2/April

Broad-Coverage Parsing with Neural Networks

Marshall R. Mayberry, III* and **Risto Miikkulainen**

Department of Computer Sciences

The University of Texas, Austin, TX 78712

martym/risto@cs.utexas.edu

Abstract

Subsymbolic systems have been successfully used to model several aspects of human language processing. Such parsers are appealing because they allow revising the interpretation as words are incrementally processed. Yet, it has been very hard to scale them up to realistic language due to training time, limited memory, and the difficulty of representing linguistic structure. In this study, we show that it is possible to keep track of long-distance dependencies and to parse into deeper structures than before based on two techniques: a localist encoding of the input sequence and a dynamic unrolling of the network according to the parse tree. With these techniques, the system can nonmonotonically parse a corpus of realistic sentences into parse trees labelled with grammatical tags from a broad-coverage Head-driven Phrase Structure Grammar of English.

Keywords: Cognitive Modeling, Natural Language Processing, Neural Networks, Parsing, Recursive Autoassociative Memory, Simple Recurrent Network

*Corresponding Author

This paper has not been submitted elsewhere in identical or similar form, nor will it be during the first three months after its submission to Neural Processing Letters.

1 Introduction

Researchers have utilized neural networks (i.e. subsymbolic) models to gain insight into human language processing. These systems develop distributed representations automatically, giving rise to a many interesting cognitive phenomena. For example, neural networks have been used to model syntactic, semantic, and thematic constraints that are seamlessly integrated to interpret linguistic data, lexical errors resulting from memory interference and overloading, aphasic and dyslexic impairments resulting from physical damage, biases, defaults and expectations that emerge from training history, as well as the robust and graceful degradation in performance that accompanies noise, damage, and incomplete or conflicting input [1, 15, 14, 16, 18, 22].

Yet, despite their many attractive characteristics, neural networks have proven very difficult to scale up to parsing realistic language. Training takes a long time, fixed-size vectors make learning long-distance dependencies difficult [2], and the linguistic formalism used imposes architectural constraints, such as the arity of parse trees. Progress has been made by introducing a number of shortcuts such as concentrating on small artificial corpora with straightforward linguistic characteristics [3, 10, 21], building in crucial linguistic heuristics such as Minimal Attachment and Right Association [11, 13], or foregoing parse trees altogether in order to concentrate on more tractable subproblems such as clause identification [9] and grammaticality judgements [12, 1, 5].

Why is subsymbolic parsing a desirable goal? The main promise for both cognitive modeling and engineering is that it accurately accounts for the nonmonotonicity of natural language processing. Over the course of the parse, the network maintains a holistic parse representation at the output. Words processed later in a sentence can change the developing representation so that the network can recover from incorrect earlier decisions. This way, the network can more effectively resolve lexical ambiguities, attachments, and anaphoric references during the course of parsing. Indeed, multiple interpretations are maintained in parallel until disambiguating information is

encountered in the input stream. This is evidently how humans process natural language, what good parsers should do, and what subsymbolic parsers promise to deliver.

The purpose of the present study is to show that deep parsing of realistic sentences is possible using neural networks: a subsymbolic neural network can be trained to read a sentence with complex grammatical structure into a distributed (holistic) representation of the parse tree. The result is based on two techniques addressing two core issues in connectionist natural language processing. The first issue is handling long-term dependencies. In earlier studies [13], we found that supplementing a Simple Recurrent Network [SRN; 7] with a self-organizing map representation of past input significantly improved performance: the network could retain the identities of words that would otherwise have been lost as the network processed new information. In this study, we show that an even simpler localist memory assembly offers sufficient improvement to allow processing long sentences.

The second issue is representing variable recurrent structure such as parse trees. In the standard approach [4, 21, 13], fixed-size representations of parse trees are built up offline from subtrees by repeated compression through Recursive Auto-Associative Memory [RAAM; 19]. The structures can then be decoded to recover the full parse tree. Unfortunately, as the depth of the tree increases, the system loses progressively more information. In this paper we propose a technique called backpropagation-through-RAAM (BPTR), similar to earlier techniques of Berg [3] and Goller and Küchler [8] developed for other reasons.

The resulting architecture, the Nonmonotonic Neural Network Parser (NNNP), combines a standard SRN with a localist memory and BPTR, and allows scaling subsymbolic parsing up to realistic sentences. This study advances the state of the art by presenting an implementation of this architecture and demonstrating that it nonmonotonically parses sentences from the CSLI Test Suite corpus from the Linguistic Grammars Online (LINGO) project at Stanford University's Center for the Study of Language and Information (CSLI). The CSLI Test Suite is designed to demonstrate the power of

Head-driven Phrase Structure Grammar [HPSG; 20]. HPSG is a constraint-based lexicalist (unification) grammar formalism that differs from more traditional derivational approaches to linguistics by emphasizing the role of the lexicon in understanding linguistic phenomena. The CSLI Test Suite demonstrates a variety of grammatical phenomena, including multiply-embedded clauses, extracted wh-words, nominal, verbal, and prepositional conjunctions, as well as compound nouns. The results suggest that holistic, nonmonotonic parsing of realistic complexity is possible to achieve in practice.

2 Representing Sequences and Structures

To process a sentence, we need to be able to represent both sequences and parse trees. Since its introduction in 1990, the Simple Recurrent Network has been widely used for sequence processing in tasks such as lexical disambiguation, prepositional phrase attachment, and active-passive transformation [4, 17, 23]. The SRN reads a sequence of input word representations into output patterns representing the parse. At each time step, a copy of the hidden layer is saved and used as input during the next step, together with the next word. In this way, each new word is interpreted in the context of the entire sequence so far, and the parse is gradually formed at the output.

One variant of the SRN uses backpropagation-through-time [BPTT; 24, 12] to improve the network's ability to process longer sequences. With BPTT, copies of the previous inputs and hidden layer activations are kept in memory, and weight updates are made with all such activations as context. The SRN is effectively trained as if it were a multi-layer feedforward network, with the constraint that the weights between each layer are shared.

In order to represent structures in a neural network, it is essential to be able to combine multiple substructures into one. The RAAM is a three-layer backpropagation network in which two or more input representations are compressed into a single representation in the hidden layer, and then decompressed back into those same inputs.

Thus, the network is trained to reproduce its input at its output. The compressed representations developed in the hidden layer are used as the input and targets for deeper compressions. For example, the representation for `[a,[b,c]]` is built up from the leaf representation for `a` and the compressed representation of `[b,c]`, which, in turn, is constructed from leaves `b` and `c` (leaf representations are usually vectors of random values). The input and hidden layers and the weights between them constitute the "encoder", while the hidden and outputs layers and their weights are the "decoder".

Both the SRN and RAAM suffer from a memory problem. As longer or deeper structures are encoded, information from earlier constituents gradually degrades to the point where it is eventually unrecoverable. The reason is that the hidden layer has a fixed size and, consequently, the more items it is forced to represent, the less accurately it is able to represent them. In the next section, we describe two techniques that help alleviate the memory degradation problem for both these networks.

3 Experiments

3.1 Network Architectures

The NNNP sentence parsing model (figure 1) consists of three essential components:

- A Simple Recurrent Network trained with BPTT.
- A localist assembly that retains a decaying activation of the input sequence.
- A dynamic unrolling of the output layer into the parse tree, trained with BPTR.

The Simple Recurrent Network trained with BPTT is the basis for the architecture, receiving distributed representations of words as input and forming compressed representations of parse trees as output.

The localist assembly is added to the SRN to solve the long-term memory problem. It is a simplification of the SARDNET self-organizing map used in earlier work [13].

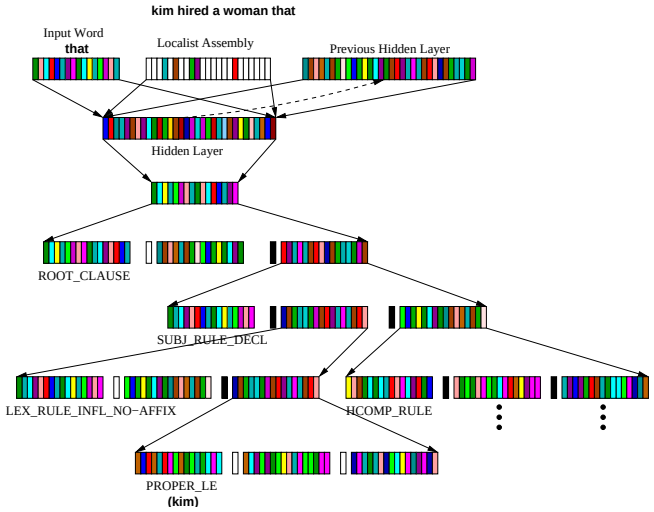

Figure 1: **The** NNNP **Architecture.** This snapshot shows the network in the middle of reading the sentence **kim hired the woman that i approved of**, together with the first four levels of the unrolled parse tree expansion (see figure 2 for the complete parse tree). The shaded units represent unit activations between 0.0 and 1.0. As in the SRN, the sentence is read one word at a time. The representation for the current input word **that** is shown at the top left. A unit corresponding to the current input word is activated on the localist assembly at a value of 1.0 and the rest of the assembly is decayed by a factor of 0.9. The network is trained with BPTT, which in effect unrolls the network in time (in the figure, only the most recent input and the previous hidden layer are shown). These inputs are propagated to the hidden layer through a matrix of weights (indicated by solid arrows) that connect all the units in one layer to those of another. The hidden layer is further propagated to the output layer through another set of weights. The output layer is dynamically unrolled into a tree of representations corresponding to the parse tree for the input sentence, with each node mapped into a label representation, a left node and a right node. The figure shows the first four levels of the parse tree up to the label matching the first input word, **kim**. The same connections are used at each level of the tree expansion. The network uses backpropagation-through-RAAM (BPTR) to update the weights in a manner analogous to the BPTT algorithm used to train the first half of the network. Note the single units preceding the left and right branches of each node in the parse tree: white indicates that the node is null, and black indicates that the node is to be expanded further. Input, output and RAAM layers had 200 units; the hidden layer had 256 units, and the localist assembly had 247 units.

As each word from the input sequence is read in, its corresponding unit in the localist assembly is activated at a value of 1.0, and the rest of the assembly decayed by a factor of 0.9. If an input word occurs more than once, the activation will already be greater than 0.0, so 1.0 is added and the current activation is taken to be the resulting average. Together with the current input and previous hidden layer, the localist assembly is used as input to the hidden layer. The localist representation identifies each input word exactly, information that could otherwise be lost in a long sequence of SRN iterations.

Retaining such information in so simple a localist assembly improves memory performance almost as much as storing it on a map, but this approach is more efficient in that it is not necessary to self-organize the map. On the other hand, it is less general in that fewer words can be represented: each unit in the SARDNET self-organizing map typically denotes several words from the lexicon. The same technique could be applied to the localist assembly, and our experience with SARDNET suggests it would have a minimal impact on performance, provided a means of allocating words reasonably uniformly to the units were used. With SARDNET, self-organization was used to allocate words to units, but the role of the map was simply to enable the underlying SRN architecture to retain past input words. This study focuses on the strongest contribution the localist assembly could make to the task of scaling up, therefore a one-to-one mapping from the lexicon is used.

The structure encoding problem is solved through Backpropagation-through-RAAM (BPTR). Whereas in BPTT, the SRN is unrolled in time, in BPTR the RAAM is unrolled to match the complete parse tree, which we term "Dynamic RAAM" to distinguish it from the procedure for training it, BPTR. Each node in the unrolled network consists of a grammatical label, a left branch, an indicator of whether that branch is null or not, and a right branch and indicator. After each word representation is read and forward-propagated through the hidden layer and through each node of the complete parse tree, the error signals from each leaf are backpropagated through the entire tree and the hidden layer. The weights are shared between levels of the parse tree and weight ad-

justments are accumulated and applied once the entire forward and backpropagation process has been completed.

The advantage of BPTR is that the network no longer has to track moving targets, which arise with standard RAAM because nonterminal representations are constantly changing as the network is trained. Second, when the SRN and RAAM are separated as in the standard approach, the errors generated by the SRN throw the RAAM off during decoding the actual representations that it receives as its input are different from what is is used to seeing during training . Integrating the SRN and RAAM networks this way eliminates the boundary between them, and the unrolled network learns to compensate for the inaccurate output of the SRN. Third, as pointed out by Berg [3], when the output layer is unrolled, it is not necessary to develop nonterminal representations offline, but they can be learned as part of the parsing process, encoding the properties of the training corpus. The learning is also sensitive to corpus statistics such as frequency of partial parse trees and lexical categories, which further improves performance.

In order to determine how each technique contributes to the system, three ablated architectures were constructed. In the first one, the localist assembly was removed from the network. In the second, a simple RAAM was substituted for the Dynamic RAAM. This ablation required significant changes to the training regime. First, the RAAM representations of the final parse trees that were to serve as targets need to be developed offline beforehand by standard RAAM training. All partial parse trees that occur in the training set needed to be identified and a separate training set from the RAAM needed to be constructed from them. Each partial parse tree was saved in a lexicon so that the SRN could be trained to output the static compressed representation of the final parse tree. The third ablated architecture had both the localist assembly and the Dynamic RAAM removed, in effect establishing a baseline comparison to the standard approach of using SRN and RAAM separately.

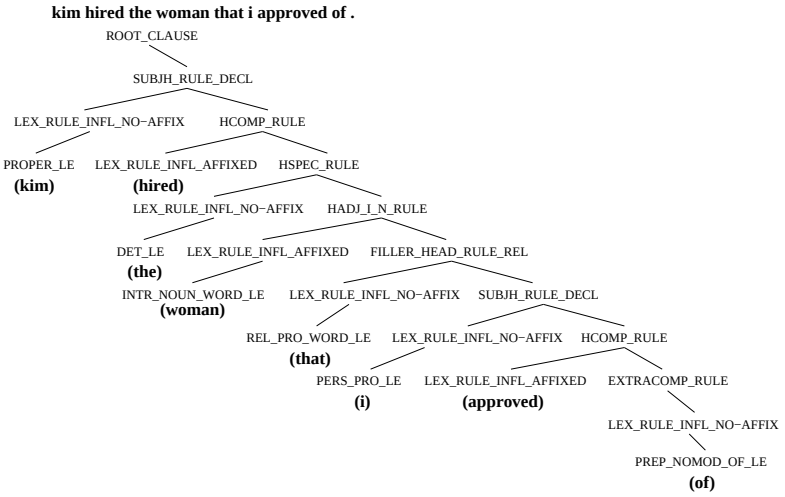

Figure 2: **HPSG Parse Tree.** This figure shows the full parse tree for the sentence **kim hired the woman that i approved of**. The terminals of the tree are lexical rules or categories for the input words, and the nonterminals are lexical or syntactic rules that are applied by the grammar to develop the parse tree. There were 172 rules and lexemes covering the corpus used in this study, many of which made such distinctions as the attachment location for an adverb or whether a form of the verb *be* is a copula or identity. An advantage of such HPSG parse trees is that they preserve the order of the words in the input sentence, as opposed to more traditional transformational grammar formalisms. Words shown in parentheses just below the leaves indicate what part of the input sentence the leaves correspond to, but are not actually part of the output.

3.2 Input Data, Training, and System Parameters

The networks were trained and tested with sentences taken from the CSLI Test Suite [6] that is distributed with release 5.2 of the English Resource Grammar (ERG), a broad-coverage Head-driven Phrase Structure Grammar of English under development for the LinGO (Linguistic Grammars Online) initiative at CSLI, Stanford University. The suite comprises 1348 sentences, both grammatical and ungrammatical, that have been chosen to demonstrate the coverage of the ERG. For this study, those 410 sentences that yield a single, unambiguous parse by the grammar were used. These sentences were expanded into 2500 sentences through interchange of nouns, pronouns, verbs, adverbs and adjectives as is commonly done in training neural networks.

Each sentence was paired with its complete syntactic HPSG parse tree, which specifies the grammatical and lexical rules that decompose the sentence into meaningful constituents and word affixes. A parse tree for a typical sentence is shown in figure 2.

Two features of HPSG parse trees are expecially worth noting. First, the trees are maximally binary; lexical rules (such as LEX_RULE_INFL_AFFIXED) and some grammatical rules (such as ROOT_CLAUSE) are unary. Second, unlike in transformational grammar, sentence word order is preserved in the HPSG parse tree. The rules themselves reflect the central role the grammatical head of a constituent plays in HPSG in constraining its complements, such as the *head-specifier rule* (HSPEC_RULE), which combines a head with its specifier. How these rules are symbolically defined is given in the ERG, but that information is not available to the NNNP parser. Rather, the network must learn to associate word sequences with the rule names as they appear in the parse tree, and generalize to new sequences.

In this study, each sentence was presented as a sequence of words and the nodes of the target parse tree were grammatical and lexical rules. Random vectors within a 200-dimensional unit hypercube were assigned to all the input words and grammatical rules and lexemes in the lexicon. Using semantic feature representations instead would likely improve the parser's performance further. However, since this study focused on memory and structure, random representations suffice. Nevertheless, a good deal of grammatical knowledge is captured by the grammar rules: for example, VP_ADV_PRE_WORD_LE versus VP_ADV_POST_WORD_LE indicates whether an adverb attaches to the beginning or to the end of a verb phrase. If the network is able to learn such distinctions with random representations, it should be able to do much better when those representations have content.

A learning curve for each architecture was obtained based on training with 20%, 40%, 60%, and 80% of the 2500 sentences in the corpus. At each percentage level, four splits of the data for training, validation, and testing were run. For each split, we randomly selected 125 sentences for a validation set. Testing was performed on the sentences that were neither in the training set nor in the validation set. Training on all 64 runs was halted when the validation error for each dataset began to level off.

Rather than use null representations for terminals, we allocated extra two "indica-

tor" units for each node in the unrolled parse tree to specify whether the node had a left, right, or both branches (figure 1). If the indicator activation is less than or equal 0.5, then the corresponding branch is null; a value greater than 0.5 indicates the branch is to be taken. This approach was found to significantly help learning.

Some rules occur several thousand times in the training data while others only occur once. Such imbalance leads the network to learn the frequent rules very well at the expense of the rare ones. To compensate, we adopted a "target radius" around each node label. If the label was output correctly within the target radius, no error for this target was backpropagated. In this way, error signals for less frequent items were backpropagated more often because the network was less likely to represent them accurately than frequent items.

All of the networks were trained with a learning rate of 0.0005, without momentum nor bias, and using the standard sum of squares cost function.

The performance of each network is measured by counting the number of terminals the network correctly identified once the entire sentence had been processed. The representation of the parse tree at the output had to be unrolled in order to determine how closely each leaf matched its target. The branch indicators complicated the measure somewhat. If they were not correct, whole subtrees of the parse tree were either inaccessible when they should have been, or bogus subtrees were accessible when they should not have been. We followed the stringent criterion of counting all rules in a branch as errors if that branch existed and that branch's indicator was below 0.5. On the other hand, a branch indicator above 0.5 with no corresponding branch in the data resulted in a single extra error. Such a simplified error is necessary because incorrect branches may sometimes never terminate.

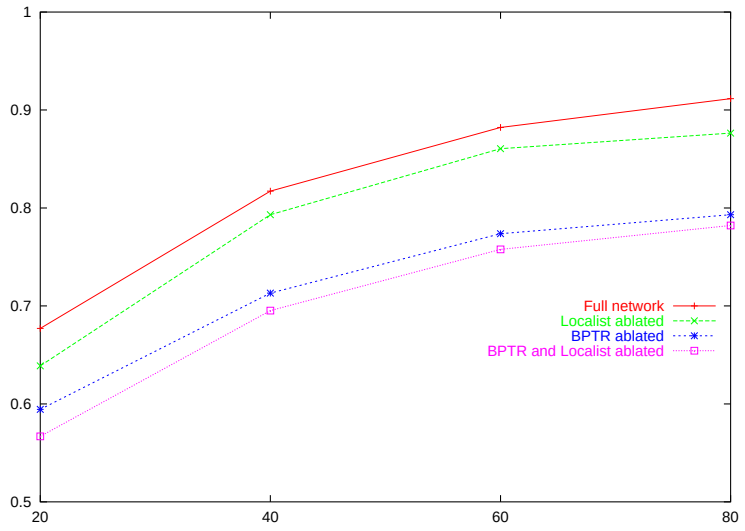

Figure 3: **Learning Curves of NNNP and its ablated versions..** The x-axis shows the percentage of the data that was in the training set, and the y-axis shows the performance of each network on the testing set averaged over four splits. The full architecture clearly outperforms the ablated networks. All of the differences are statistically significant with $p < 0.05$.

4 Results

The main result is that both BPTR and the localist assembly substantially improve performance over the standard SRN and RAAM approach (figure 3). BPTR contributes most of this improvement; when the localist assembly is ablated, the performance drops slightly less, although still significantly. Training the full and localist-assembly-ablated architectures took over a month on a 500 MHz Pentium III workstation on the 80% dataset, and proportionally less on the smaller datasets. The performance curves for these architectures started leveling off around 1000 epochs. The BPTR-ablated architectures had to be trained in two phases; the offline RAAM training took less than a week, and the parser that used the RAAM targets took another two weeks. Although faster per epoch than the networks with BPTR, the BPTR-ablated networks did not begin to level off until close to 2000 epochs.

We have shown that the parser can process realistic language. To be a valid cognitive model, it must also be able to parse nonmonotonically. We show how a typical test

sentence is processed to see how the network is able to recover from early decisions.

In the example below, the numbers below each word indicate the number of mismatches in the parse tree represented in the output as each word is read in:

kim	hired	the	woman	that	i	approved	of	.
8	4	3	5	3	2	0	0	0

When the first word, **kim**, is read in, the network's output represents a very different sentence structure than its target parse tree, resulting in a mismatch of eight grammatical labels. If we examine the training dataset, we find that the network's output is closest to the parse tree for sentences like **kim got hired**, because sentences with this structure are more numerous than others beginning with **kim**. Upon reading **hired**, the network's output decodes into a parse tree that is now closer to the target: that of the sentence **kim hired a woman that was competent**, which involves an embedded relative clause like the test sentence. The next word, **the**, does not change the output significantly, but just corrects the article. When **woman** is read in, the output decodes into a parse tree for a sentence like **kim hired the woman sara approved of**. Notice that this sentence lacks the relative pronoun **that**, which accounts for the extra mismatches. Upon reading **that** next, the parse tree matches sentences like **kim hired the woman that sara approved of**, which are more frequent in the training dataset than those with a pronoun (such as **i**) in the relative clause. The pronoun **i** is next read, but this time the output decodes into a parse tree for a sentence like **kim hired the woman that i interviewed**, since, again, this particular construction is more frequent in the training set. When **approved** is reached, the output finally decodes into the correct parse tree, which does not occur in the training set. It is this nonmonotonic ability to recover from early mistakes and the ability to generalize that makes subsymbolic parsers particularly intriguing.

The nonmonotonic parsing ability derives from the same attractor dynamics mechanism that makes recurrent neural networks characteristically robust. In an informal experiment the NNNP architecture was found able to tolerate both the random insertion

of a new word in the sentence (e.g., to model dysfluencies) and up to approximately 12% noise added to the network's weights without a significant degradation in performance. While neural networks have been shown to perform robustly in numerous earlier studies, nonmonotonic broad-coverage parsing is a novel and equally important cognitively plausible behavior.

5 Discussion and Future Work

The ultimate goal of this research is to develop a subsymbolic parser that can handle realistic language without sacrificing those characteristics of neural networks that make them powerful cognitive models. Distributed representation of parse trees permits the network to gradually refine its output to accommodate changes as new information comes in. In this paper, we have shown that this behavior can be preserved while scaling up to realistic linguistic structures.

The scale-up is achieved through two techniques: localist input assembly and BPTR. The localist assembly allows the network to keep track of long-term dependencies so that the correct rule labels in the parse tree can be maintained. It does so by identifying the constituents exactly, without the gradual blurring that happens in the standard SRN.

Integrating the Dynamic RAAM into the parser reduces the memory bottleneck problem of standard RAAM. In BPTR, the network develops compressed representations that facilitate mapping the input sequence to its parse tree. This technique is similar to the one used by Berg [3], as well as to Backprop-through-Structure [BPTS; 8], although there are some differences. Berg's XERIC parser used a five-layer network based on Sequential RAAM [SRAAM; 19], whereas NNNP is based on the SRN. The sentences used in Berg's study were relatively simple, and did not include, for example, embedded clauses nor conjunctions. Our study extends that of Berg in showing that the technique will scale up to realistic sentences. Goller and Küchler's BPTS was developed for an entirely different task, i.e., classifying logical terms. BPTS was only

applied to the encoder part of the network, whereas BPTR is applied to the decoder.

The network performs quite well even with random vectors for words and rules. It would be interesting to have the network develop its own input and output representations as part of the learning [possibly with FGREP; 14]. This way, the lexical items would be optimized according to how phrases are parsed and like the nonterminal representations in BPTR. The word representations would begin to embody the contexts in which they occurred to minimize the error between what the word is meant to be, and what the output is. This process should improve the parser's performance even further.

6 Conclusion

In this study we demonstrated two techniques, localist input and BPTR, that alleviate the memory and structure learning bottlenecks in subsymbolic NLP. These approaches permit scaling up to real-world language by training parsers on a corpus of realistic structures. As is typical of holistic parsers, the parse result develops nonmonotonically while an input sentence is read, making the system an appealing cognitive model.

References

[1] Joseph Allen and Mark S. Seidenberg. The emergence of grammaticality in connectionist networks. In B. MacWhinney, editor, *Emergence of Language*, pp. 115–151. Erlbaum, Hillsdale, NJ, 1999.

[2] Yoshua Bengio, Patrick Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[3] George Berg. A connectionist parser with recursive sentence structure and lexical disambiguation. In W. Swartout, editor, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pp. 32–37. Cambridge, MA: MIT Press, 1992.

[4] David J. Chalmers. Syntactic transformations on distributed representations. *Connection Science*, 2:53–62, 1990.

[5] Morten H. Christiansen and Nick Chater. Toward a connectionist model of recursion in human linguistic performance. *Cognitive Science*, 23(2):157–205, 1999.

[6] Ann Copestake, John Carroll, Rob Malouf, and Stephan Oepen. *The (New) LKB System*. CSLI, Stanford University, 1999.

[7] Jeffrey Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[8] C. Goller and A. Küchler. Learning task-dependent distributed representations by backpropagation through structure. *IEEE Transactions on Neural Networks*, 1:347–352, 1996.

[9] James Hammerton. Clause identification with long short-term memory. In W. Daelemans and R. Zajac, editors, *Proceedings of CoNLL-2001*, pp. 61–63, 2001.

[10] Edward Kei Shiu Ho and Lai Wan Chan. Analyzing holistic parsers: Implications for robust parsing and systematicity. *Neural Computation*, 13(5):1137–1170, 2001.

[11] Peter C. R. Lane and James B. Henderson. Incremental syntactic parsing of natural language corpora with simple synchrony networks. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):219–231, 2001.

[12] Steve Lawrence, C. Lee Giles, and Sandiway Fong. Natural language grammatical inference with recurrent neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 12(1):126–140, 2000.

[13] Marshall R. Mayberry, III and Risto Miikkulainen. Combining maps and distributed representations for shift-reduce parsing. In S. Wermter and R. Sun, editors, *Hybrid Neural Systems*, pp. 144–157. Springer-Verlag, 2000.

[14] Risto Miikkulainen. *Subsymbolic Natural Language Processing: An Integrated Model of Scripts, Lexicon, and Memory*. MIT Press, Cambridge, MA, 1993.

[15] Risto Miikkulainen. Dyslexic and category-specific impairments in a self-organizing feature map model of the lexicon. *Brain and Language*, 59:334–366,

1997.

[16] William C. Morris, Garrison W. Cottrell, and Jeffrey L. Elman. A connectionist simulation of the empirical acquisition of grammatical relations. volume 1778, pp. 175–193, Berlin; New York, 2000. Springer-Verlag.

[17] Paul Munro, Cynthia Cosic, and Mary Tabasko. A network for encoding, decoding and translating locative prepositions. *Connection Science*, 3:225–240, 1991.

[18] David C. Plaut and Tim Shallice. Perseverative and semantic influences on visual object naming errors in optic aphasia: A connectionist account. *Journal of Cognitive Neuroscience*, 5(1):89–117, 1993.

[19] Jordan B. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46:77–105, 1990.

[20] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. Studies in Contemporary Linguistics. The University of Chicago Press, Chicago, IL, 1994.

[21] Noel E. Sharkey and Amanda J. C. Sharkey. A modular design for connectionist parsing. In M. Drossaers and A. Nijholt, editors, *Twente Workshop on Language Technology 3: Connectionism and Natural Language Processing*, pp. 87–96, Enschede, the Netherlands, 1992. Department of Computer Science, University of Twente.

[22] Mark F. St. John and James L. McClelland. Learning and applying contextual constraints in sentence comprehension. *Artificial Intelligence*, 46:217–258, 1990.

[23] David S. Touretzky. Connectionism and compositional semantics. In J. Barnden and J. Pollack, editors, *High-Level Connectionist Models*, volume 1 of *Advances in Connectionist and Neural Computation Theory,* Barnden, J. A., series editor, pp. 17–31. Ablex, Norwood, NJ, 1991.

[24] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280, 1989.