SARDSRN: A Neural Network Shift-Reduce Parser

Marshall R. Mayberry, III
Department of Computer Sciences
The University of Texas
Austin, TX, 78712, USA
martym@cs.utexas.edu

Risto Miikkulainen
Department of Computer Sciences
The University of Texas
Austin, TX, 78712, USA
risto@cs.utexas.edu

Technical Report AI98-275
Septermber 1998

Abstract

Simple Recurrent Networks (SRNs) have been widely used in natural language tasks. SARDSRN extends the SRN by explicitly representing the input sequence in a SARDNET self-organizing map. The distributed SRN component leads to good generalization and robust cognitive properties, whereas the SARDNET map provides exact representations of the sentence constituents. This combination allows SARDSRN to learn to parse sentences with more complicated structure than can the SRN alone, and suggests that the approach could scale up to realistic natural language.

1 Introduction

The subsymbolic approach (i.e. neural networks with distributed representations) to processing language is attractive for several reasons. First, it is inherently robust: the distributed representations display graceful degradation of performance in the presence of noise, damage, and incomplete or conflicting input (Miikkulainen 1993; St. John and McClelland 1990). Second, because computation in these networks is constraint-based, the subsymbolic approach naturally combines syntactic, semantic, and thematic constraints on the interpretation of linguistic data (McClelland and Kawamoto 1986). Third, subsymbolic systems can be lesioned in various ways and the resulting behavior is often strikingly similar to human impairments (Miikkulainen 1993, 1996; Plaut 1991). These properties of subsymbolic systems have attracted many researchers in the hope of accounting for interesting cognitive phenomena, such as role-binding and lexical errors resulting from memory interference and overloading, aphasic and dyslexic impairments resulting from physical damage, and biases, defaults and expectations emerging from training history (Miikkulainen 1997, 1996, 1993)

Since its introduction in 1990, the simple recurrent network (SRN; Elman 1990) has become a mainstay in connectionist natural language processing tasks such as lexical disambiguation, prepositional phrase attachment, active-passive transformation, anaphora resolution, and translation (Allen 1987; Chalmers 1990; Munro et al. 1991; Touretzky 1991). This paper describes an extension to the standard SRN, which utilizes SARDNET (James and Miikkulainen 1995), a self-organizing map algorithm designed to represent sequences. SARDNET permits the sequence information to remain explicit, yet distributed in the sense that similar sequences result in similar patterns on the map. SARDSRN, the combination of the SRN and SARDNET, effectively solves the fundamental memory accuracy limitations of the SRN, and allows the processing of sentences of realistic length.

This paper shows how SARDSRN improves upon the performance of the SRN in a nontrivial syntactic shift-reduce parsing task. The results show that SARDSRN outperforms the SRN in this task by providing an effective solution to the memory problem. SARDSRN therefore forms a solid foundation for building a subsymbolic parser of realistic language.

2 The Task: Shift-Reduce Parsing

The task taken up in this study, shift-reduce (SR) parsing, is one of the simplest approaches to sentence processing that nevertheless has the potential to handle a substantial subset of English (Marcus 1980; Tomita 1986). Its basic formulation is based on the pushdown automata for parsing context-free grammars, but it can be extended to context-sensitive grammars as well.

The parser consists of two data structures: the input buffer stores the sequence of words remaining to be read, and the partial parse results are kept on the stack (figure 1). Initially the stack is empty and the entire sentence is in the input buffer. At each step, the parser has to decide whether to shift a word from the buffer to the stack, or to reduce one or more of the top elements of the stack into a new element representing their combination. For example, if the top two elements are currently *NP* and *VP*, the parser reduces them into *S*, corresponding to the grammar rule $S \rightarrow NP\ VP$ (step 17 in figure 1). The process stops when the elements in the stack have been reduced to S, and no more words remain in the input. The reduce actions performed by the parser in this process constitute the parse result, such as the syntactic parse tree (line 18 in figure 1).

The sequential scanning process and incremental forming of partial representations is a plausible cognitive model for language understanding. SR parsing is also very efficient, and lends itself to many extensions. For example, the parse rules can be made more context sensitive by taking more of the stack and the input buffer into account. Also, the partial parse results may consist of syntactic or semantic structures.

The general SR model can be implemented in many ways. A set of symbolic shift-reduce rules can be written by hand or learned from input examples (Hermjacob and Mooney 1997; Simmons and Yu 1991; Zelle and Mooney 1993, 1996). It is also possible to train a neural network to make shift/reduce decisions, based on the current stack and the input buffer as input. If trained properly, the neural network can generalize well to new sentences (Simmons and Yu 1992). Whatever correlations there exist between the word representations

Stack	Input Buffer	Action	
	the boy who liked the girl chased the cat .	1	Shift
the	boy who liked the girl chased the cat .	2	Shift
the boy	who liked the girl chased the cat .	3	Reduce
NP[the,boy]	who liked the girl chased the cat .	4	Shift
NP[the,boy] who	liked the girl chased the cat .	5	Shift
NP[the,boy] who liked	the girl chased the cat .	6	Shift
NP[the,boy] who liked the	girl chased the cat .	7	Shift
NP[the,boy] who liked the girl	chased the cat .	8	Reduce
NP[the,boy] who liked NP[the,girl]	chased the cat .	9	Reduce
NP[the,boy] who VP[liked,NP[the,girl]]	chased the cat .	10	Reduce
NP[the,boy] RC[who,VP[liked,NP[the,girl]]]	chased the cat .	11	Reduce
NP[NP[the,boy],RC[who,VP[liked,NP[the,girl]]]]	chased the cat .	12	Shift
NP[NP[the,boy],RC[who,VP[liked,NP[the,girl]]]] chased	the cat .	13	Shift
NP[NP[the,boy],RC[who,VP[liked,NP[the,girl]]]] chased the	cat .	14	Shift
NP[NP[the,boy],RC[who,VP[liked,NP[the,girl]]]] chased the cat	.	15	Reduce
NP[NP[the,boy],RC[who,VP[liked,NP[the,girl]]]] chased NP[the,cat]	.	16	Reduce
NP[NP[the,boy],RC[who,VP[liked,NP[the,girl]]]] VP[chased,NP[the,cat]]	.	17	Reduce
S[NP[NP[the,boy],RC[who,VP[liked,NP[the,girl]]]],VP[chased,NP[the,cat]]]		18	Stop

Figure 1: **Shift-Reduce Parsing a Sentence.** Each step in the parse is represented by a line from top to bottom. The current stack is at left, the input buffer in the middle, and the parsing decision in the current situation at right. At each step, the parser either shifts a word into the stack, or reduces the top elements of the stack into a higher-level representation, such as `the boy` $\rightarrow$ `NP[the,boy]` (step 3).

and the appropriate parsing decisions, the network will learn to utilize them.

Another important extension is to implement the stack as a neural network. This way the parser can have access to the entire stack at once, and interesting cognitive phenomena in processing complex sentences can be modeled. The SPEC system (Miikkulainen 1996) was a first step in this direction. The stack was represented as a compressed distributed representation, formed by a RAAM auto-encoding network (Recursive Auto-Associative Memory; Pollack 1990). The resulting system was able to parse complex relative clause structures. When the stack representation was artificially lesioned by adding noise, the parser exhibited very plausible cognitive performance. Shallow center embeddings were easier to process, as were sentences with strong semantic constraints in the role bindings. When the parser made errors, it usually switched the roles of two words in the sentence, which is what people also do in similar situations. A symbolic representation of the stack would make modeling such behavior very difficult.

The SPEC architecture, however, was not a complete implementation of SR parsing; it was designed specifically for embedded relative clauses. For general parsing, the SR stack needs to be encoded with neural networks, to make it possible to parse much more varied linguistic structures. We believe that the generalization and robustness of subsymbolic neural networks will result in powerful, cognitively valid performance. However, the main problem of limited memory accuracy of the SRN parsing network must first be solved. An architecture that will do that, SARDSRN, will be described next.

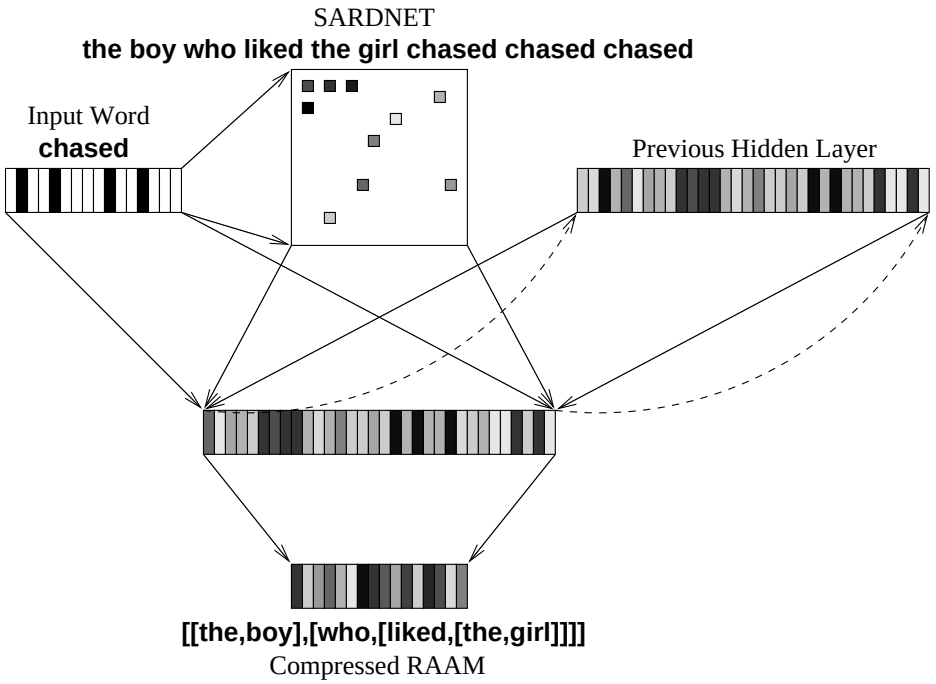

Figure 2: **The SARDSRN Network.** This snapshot shows the network during step 11 of figure 1. The representation for the current input word, `chased`, is shown at top left. Each word is input to the SARDNET map, which builds a representation for the sequence word by word. At each step, the previous activation of the hidden layer is copied (as indicated by the dotted line) to the Previous Hidden Layer assembly. This activation, together with the current input word and the current SARDNET pattern, is propagated to the hidden layer of the SRN network. As output, the network generates the compressed RAAM representation of the top element in the shift-reduce stack at this state of the parse (in this case, line 12 in figure 1). SARDNET is a map of word representations, and it is trained through the Self-Organizing Map algorithm (SOM; Kohonen 1997, 1990). All other connections are trained through backpropagation.

3 The SARDSRN parser architecture

3.1 Simple Recurrent Network

The starting point for SARDSRN (figure 2) is the simple recurrent network. The network reads a sequence of input word representations into output patterns representing the parse results, such as syntactic or case-role assignments for the words. 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 result is gradually formed at the output.

The SRN architecture can be used to implement a shift-reduce parser in the following way: the network is trained to step through the parse (such as that in figure 1), generating a compressed distributed representation

of the top element of the stack at each step (formed by a RAAM network: section 4.1). The network reads the sequence of words one word at a time, and each time either shifts the word onto the stack (by passing it through the network, e.g. step 1), or performs one or more reduce operations (by generating a sequence of compressed representations corresponding to the top element of the stack: e.g. steps 8-11). After the whole sequence is input, the final stack representation is decoded into a parse result such as a parse tree. Such an architecture is powerful for two reasons: (1) During the parse, the network does not have to guess what is coming up later in the sentence, as it would if it always had to shoot for the final parse result; its only task is to build a representation of the current stack in its hidden layer and the top element in its output. (2) Instead of having to generate a large number of different stack states at the output, it only needs to output representations for a relatively small number of common substructures. Both of these features make learning and generalization easier.

A well-known problem with the SRN model is its low memory accuracy. It is difficult for it to remember items that occurred several steps earlier in the input sequence, especially if the network is not required to produce them in the output layer during the intervening steps (Stolcke 1990; Miikkulainen 1996). The intervening items are superimposed in the hidden layer, obscuring the traces of earlier items. Nor has simply increasing the size of the hidden layer been found to offer much advantage. As a result, parsing with an SRN has been limited to relatively simple sentences with shallow structure.

3.2 SARDNET

The solution described in this paper is to use an explicit representation of the input sequence as additional input to the hidden layer. This representation provides more accurate information about the sequence, such as the relative ordering of the incoming words, and it can be combined with the weak hidden layer representation to generate accurate output that retains all the advantages of distributed representations. The sequence representation must be explicit enough to allow such cleanup, but it must also be compact and generalize well to new sequences.

The SARDNET (Sequential Activation Retention and Decay NETwork; James and Miikkulainen 1995) self-organizing map for sequences has exactly these properties. SARDNET is based on the Self-Organizing Map neural network (SOM; Kohonen 1990, 1997), and organized to represent the space of all possible word representations. As in a conventional self-organizing map network, each input word is mapped onto a particular map node called the maximally-responding unit, or winner. The weights of the winning unit and all the nodes in its neighborhood are updated according to the standard adaptation rule to better approximate the current input. The size of the neighborhood is set at the beginning of the training and reduced as the map becomes more organized.

In SARDNET, the sequence of words is represented as a distributed activation pattern on the map (figure 2). For each word, the maximally responding unit is activated to a maximum value of 1.0, and the activations of units representing previous words are decayed according to a specified decay rate (e.g. 0.9). After a unit is activated, it is removed from competition and cannot represent later words in the sequence. In this manner

Noun(0) → **boy**	*Noun*(1) → **girl**	*Noun*(2) → **dog**	*Noun*(3) → **cat**
Verb(0,0) → **liked**, **saw**	*Verb*(0,1) → **liked**, **saw**		*Verb*(0,2) → **liked**
Verb(0,3) → **chased**	*Verb*(1,0) → **liked**, **saw**		*Verb*(1,1) → **liked**, **saw**
Verb(1,2) → **liked**	*Verb*(1,3) → **chased**		*Verb*(2,0) → **bit**
Verb(2,1) → **bit**	*Verb*(2,2) → **bit**		*Verb*(2,3) → **bit**, **chased**
Verb(3,0) → **saw**	*Verb*(3,1) → **saw**		*Verb*(3,3) → **chased**
S → *NP*(n) *VP*(n,m)	*VP*(n,m) → *Verb*(n,m) *NP*(m)		*NP*(n) → *Noun*(n)
RC(n) → **who** *VP*(n,m)	*NP*(n) → *Noun*(n) *RC*(n)		*RC*(n) → **whom** *NP*(m) *Verb*(m,n)

Figure 3: **Grammar.** This phrase structure grammar generates sentences with subject- and object-extracted relative clauses. Agreement between subject and object depend on the verb in the clause as encoded in the rule arguments.

each unit may represent different words depending on the context, which allows for an efficient representation of sequences, and also generalizes well to new sequences.

In the SARDSRN architecture, a SARDNET representation of the input word sequence is formed at the same time as the SRN hidden layer representation, and used together with the previous hidden layer representation and the next input word as input to the hidden layer (figure 2). This architecture allows the SRN to perform its task with significantly less memory degradation. The sequence information remains accessible in SARDNET, and the SRN is able to focus on capturing correlations relating to sentence constituent structure during parsing.

4 Experiments

4.1 Input Data, Training, and System Parameters

The data used to train and test the SRN and SARDSRN networks were generated from the phrase structure grammar in figure 3, adapted from a grammar that has become common in the literature (Elman 1991; Miikkulainen 1996). Since our focus was on shift-reduce parsing, and not processing relative clauses per se, sentence structure was limited to one relative clause per sentence. From this grammar training targets corresponding to each step in the parsing process were obtained. For shifts, the target is simply the current input. In these cases, the network is trained to auto-associate, which these networks are good at. For reductions, however, the targets consist of representations of the partial parse trees that result from applying a grammatical rule. For example, the reduction of the sentence fragment `who liked the girl` would produce the partial parse result `RC[who,VP[liked,NP[the,girl]]]`. Two issues arise: how should the parse trees be represented, and how should reductions be processed during sentence parsing?

The approach taken in this paper is the same as in SPEC (section 2), as well as in other connectionist parsing systems (Miikkulainen 1996; Berg 1992; Sharkey and Sharkey 1992). Compressed representations of the syntactic parse trees using RAAM are built up through auto-association of the constituents. This training is performed beforehand separately from the parsing task. Once formed, the compressed representations can be decoded into their constituents using just the decoder portion of the RAAM architecture.

the	10000000	who	01010000
whom	01100000	.	11111111
boy	00101000	dog	00100010
girl	00100100	cat	00100001
chased	00011000	saw	00010010
liked	00010100	bit	00010001

Figure 4: **Lexicon.** Each word representation is put together from a part-of-speech identifier (first four components) and a unique ID tag make up (last four). This encoding is then repeated eight times to form a 64-unit word representation. Such redundancy makes it easier to identify the word.

In shift-reduce parsing, the input buffer after each "Reduce" action is unchanged; rather, the reduction occurs on the stack. Therefore, if we want to perform the reductions one step at a time, the current word must be maintained in the input buffer until the next "Shift" action. Therefore, the input to the network consists of the sequence of words that make up the sentence with the input word repeated for each reduce action, and the target consists of representations of the top element of the stack (as shown in figure 1).

Word representations were hand-coded to provide basic part-of-speech information together with a unique ID tag that identified the word within the syntactic category (figure 4). The basic encoding of 8 units was repeated eight times to form a 64-unit representation. Such redundant long representations were found to facilitate learning in general.

Four data sets of 20%, 40%, 60%, and 80% of the 436 sentences generated by the grammar were randomly selected to train both parsers, and each parser was trained on each dataset four times. Training on all thirty-two runs was stopped when the error on a 22-sentence (5%) validation set began to level off. The same validation set was used for all the simulations and was randomly drawn from a pool of sentences that did not appear in any of the training sets. Testing was then performed on the remaining sentences that were neither in the training set nor in the validation set.

The SRN network architecture consisted of a 64-unit input layer, 200-unit hidden and context layers, and 64-unit output and target layers. SARDSRN added a 144-unit feature map (SARDNET) to the SRN setup. A learning rate of 0.2 was used to train both networks, while the learning and decay rates for the SARDNET feature map input in SARDSRN were set to 0.8 and 0.9, respectively. The neighborhood was set at 5 initially and gradually reduced to 0. These parameters were found experimentally to result in the best general performance for both parsers.

4.2 Results

Three different performance measures were used to obtain a thorough characterization of the relative performances of the SRN and SARDSRN architectures.

The first measure, the *epoch error*, is the average error per output unit during each epoch. This measure

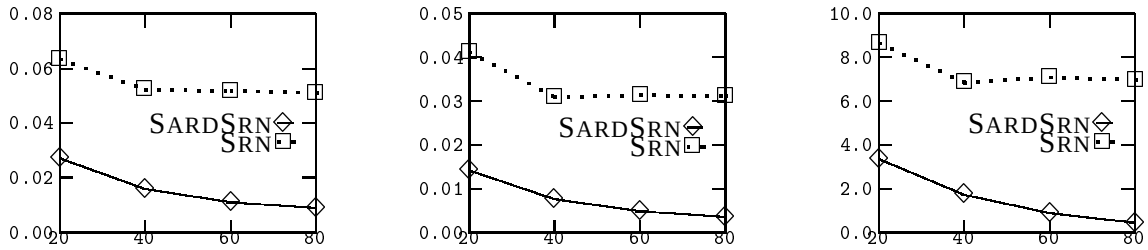

Figure 5: **Results.** Averages over four simulation runs using the three performance measures, *epoch error* (left), *leaf error* (middle), and *average mismatches* (right) on the test data. The graphs all show the same result: the SRN's performance in all 16 runs bottomed out at a much higher error than SARDSRN, while it was still unable to parse all of the training and test sentences. SARDSRN, on the other hand, did learn to parse the training sentences, and showed very good generalization to the test sentences. These differences are statistically significant with $p < 0.00005$.

tells us how closely the output representations matched the target representations during parsing. Presumably, if the epoch error is low, the output representations still permit accurate decoding into the correct parse tree. However, because this measure only reports the average performance over an entire epoch, it gives us no sense of the network's performance at each step in the parsing process. For example, there remains the danger that a low epoch error could also be achieved by learning the shift operations very accurately, with lower accuracy on the reductions, resulting in an incorrect decoding of the compressed representations of the parse tree.

The second measure, the *leaf error*, applies the first measure on the leaves of the RAAM representations. Each RAAM representation encountered during parsing was decoded and the errors at the leaves accumulated and averaged. The leaf error tends to be lower than the epoch error because the leaves are always binary representations, which are easier for the networks to get right than the continuous representations (Noelle et al. 1997). Because the identity of the word representations depends on only a few units (the ID tag: see figure 4), it is possible to generate low error over the part-of-speech part of the representation without correctly identifying the word.

The final measure, *average mismatches*, therefore, reports the average number of leaf representations that could not be correctly identified by nearest match in Euclidean distance from the lexicon. As an example, if the target is `RC[who,VP[liked,NP[the,girl]]]`, (step 11 of figure 1), but the network output is `RC[who,VP[saw,NP[the,girl]]]`, then a mismatch would occur at the leaf labelled `saw` once the RAAM representation was decoded. Average mismatches provide a measure of the correctness and utility of the information in the RAAM representation.

Training took about four days on a 200 MHz Pentium Pro workstation, with SARDSRN taking about 1.5 times as long per epoch as the SRN alone. The validation error in the SRN runs quickly leveled off, and continued training did nothing to improve it. On the other hand, the SARDSRN simulation runs were still showing slight improvements when they were cut off. Figure 5 plots these performance measures averaged over the four simulation runs against the test sentences.

By all measures, SARDSRN performed significantly–even qualitatively–better than the standard SRN. On

the training datasets, there was roughly an order of magnitude difference in both the epoch errors and the leaf errors between SARDSRN and SRN, and at least two orders of magnitude difference in the average number of mismatches per sentence. These results suggest that the SRN could not even learn the training data to any useful extent, whereas SARDSRN does not appear to be nearing its limit. On the test sets, the epoch error and leaf error for the SRN never fell below 0.05 and 0.03, respectively, and there were nearly 7 mismatches per sentence on average. Even in the most difficult case for the SARDSRN (on the 20% test dataset, in which the networks were trained on just 89 sentences, and tested on 325), these errors never reached half that level. These results show that SARDSRN forms a promising starting point for parsing sentences of realistic length and complexity.

4.3 Example Parse

Adding SARDNET to the SRN architecture made it possible for the network to learn the parsing task. This can be shown clearly by contrasting the performances of SARDSRN and the SRN on a typical sentence, such as the one in figure 1. Neither SARDSRN nor SRN had any trouble with the shift targets. Not surprisingly, early in training the networks would master all the shift targets in the sentence before they would get any of the reductions correct. The first reduction (`NP[the,boy]` in our example) also poses no problem for either network. Nor, in general, does the second, `NP[the,girl]`, because the constituent information is still fresh in memory. However, the ability of the SRN to generate the later reductions accurately degrades rapidly because the information about earlier constituents is smothered by the later steps of the parse. Interestingly, the structural information survives much longer. For example, instead of `RC[who,VP[liked,NP[the,girl]]]`, the SRN might produce `RC[who,VP[bit,NP[the,dog]]]`. The structure of this representation is correct; what is lost are the particular instantiations of the parse tree. This is where SARDNET makes a difference. The lost constituent information remains accessible in the feature map. As a result, SARDSRN is able to capture each constituent even through the final reductions.

5 Discussion

These results demonstrate a practicable solution to the memory degradation problem of simple recurrent networks. The SRN does not have to maintain specific information about the sequence constituents, and can instead focus on what it is best at: *capturing structure*. Although the sentences used in these experiments are still relatively uncomplicated, they do exhibit enough structure to suggest that much more complex sentences could be tackled with SARDSRN.

The operation of SARDSRN on the shift-reduce parsing task is a nice demonstration of holistic computation. The network is able to learn how to generate each RAAM parse representation during the course of sentence processing without ever having to decompose and recompose the constituent representations. Partial parse results can be built up incrementally into increasingly complicated structures, which suggests that training could be performed incrementally. Such a training scheme is especially attractive given that training

in general is still relatively costly.

An important extension of the SARDSRN idea, currently being investigated by our group, is an architecture where SARDNET is combined with a RAAM network. RAAM, although having many desirable properties for a purely connectionist approach to parsing, has long been a bottleneck during training. Its operation is very similar to the SRN, and it suffers from the same memory accuracy problem: with deep structures the superimposition of higher-level representations gradually obscure the traces of low-level items, and the decoding becomes inaccurate. This degradation makes it difficult to use RAAM to encode/decode parse results of realistic language. Our expectation is that the explicit representation of a compressed structure formed on a SARDNET feature map, when coupled with the distributed representations of the RAAM, will result in an architecture able to represent much richer linguistic structure.

The SARDSRN idea is not just a way to improve the performance of subsymbolic networks; it is an explicit implementation of the idea that humans can keep track of identities of elements, not just their statistical properties (Miikkulainen 1993). The subsymbolic networks are very good with statistical associations, but cannot distinguish between representations that have similar statistical properties. People can; whether they use a map-like representation is an open question, but we believe the SARDNET representation suggests a way to capture a lot of the resulting behavior. It is useful for building powerful subsymbolic language understanding systems, but it is also a plausible cognitive approach.

6 Conclusion

We have described an extension of the SRN called SARDSRN that combines the subsymbolic distributed properties of the SRN with the localist properties of SARDNET. The distributed component leads to good generalization and robust cognitive properties, whereas the map provides exact representations of the sentence constituents. The results in this paper demonstrate a practicable solution to the memory degradation problem of SRNs. With SARDNET keeping track of the sequence constituents, the SRN is able to learn the structure representation necessary to perform shift-reduce parsing. This combination allows SARDSRN to learn to parse longer and more complex sentences than the SRN alone. Future research will focus on extending this approach to the RAAM architecture, with the expectation that the representative properties of SARDNET will allow RAAM to encode deeper structures, such as the feature-value matrices used in the lexicalist, constraint-based grammar formalisms of contemporary linguistics theory, that are a prerequisite to handling realistic natural language.

Acknowledgments

This research was supported in part by the Texas Higher Education Coordinating Board under grant ARP-444.

References

Allen, R. B. (1987). Several studies on natural language and back-propagation. In *Proceedings of the IEEE First International Conference on Neural Networks* (San Diego, CA), vol. II, 335–341. Piscataway, NJ: IEEE.

Berg, G. (1992). A connectionist parser with recursive sentence structure and lexical disambiguation. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 32–37. Cambridge, MA: MIT Press.

Chalmers, D. J. (1990). Syntactic transformations on distributed representations. *Connection Science*, 2:53–62.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14:179–211.

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

Hermjacob, U., and Mooney, R. J. (1997). Learning parse and translation decisions from examples with rich context. In *Proceedings of the 35th Annual Meeting of the ACL*.

James, D. L., and Miikkulainen, R. (1995). SARDNET: A self-organizing feature map for sequences. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems 7*, 577–584. Cambridge, MA: MIT Press.

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, 78:1464–1480.

Kohonen, T. (1997). *Self-Organizing Maps*. Berlin; New York: Springer. Second edition.

Marcus, M. (1980). *A Theory of Syntactic Recognition for Natural Language*. Cambridge, MA: MIT Press.

McClelland, J. L., and Kawamoto, A. H. (1986). Mechanisms of sentence processing: Assigning roles to constituents. In McClelland, J. L., and Rumelhart, D. E., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 2: Psychological and Biological Models*, 272–325. Cambridge, MA: MIT Press.

Miikkulainen, R. (1993). *Subsymbolic Natural Language Processing: An Integrated Model of Scripts, Lexicon, and Memory*. Cambridge, MA: MIT Press.

Miikkulainen, R. (1996). Subsymbolic case-role analysis of sentences with embedded clauses. *Cognitive Science*, 20:47–73.

Miikkulainen, R. (1997). Dyslexic and category-specific impairments in a self-organizing feature map model of the lexicon. *Brain and Language*, 59:334–366.

Munro, P., Cosic, C., and Tabasko, M. (1991). A network for encoding, decoding and translating locative prepositions. *Connection Science*, 3:225–240.

Noelle, D. C., Cottrell, G., and Wilms, F. (1997). Extreme attraction: The benefits of corner attractors. Technical Report CS97-536, Department of Computer Science and Engineering, UCSD, San Diego, CA.

Plaut, D. C. (1991). *Connectionist Neuropsychology: The Breakdown and Recovery of Behavior in Lesioned Attractor Networks*. PhD thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA. Technical Report CMU-CS-91-185.

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence*, 46:77–105.

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

Simmons, R. F., and Yu, Y.-H. (1991). The acquisition and application of context sensitive grammar for English. In *Proceedings of the 29th Annual Meeting of the ACL*. Morristown, NJ: Association for Computational Linguistics.

Simmons, R. F., and Yu, Y.-H. (1992). The acquisition and use of context dependent grammars for English. *Computational Linguistics*, 18:391–418.

St. John, M. F., and McClelland, J. L. (1990). Learning and applying contextual constraints in sentence comprehension. *Artificial Intelligence*, 46:217–258.

Stolcke, A. (1990). Learning feature-based semantics with simple recurrent networks. Technical Report TR-90-015, International Computer Science Institute, Berkeley, CA.

Tomita, M. (1986). *Efficient Parsing for Natural Language*. Dordrecht; Boston: Kluwer.

Touretzky, D. S. (1991). Connectionism and compositional semantics. In Barnden, J. A., and Pollack, J. B., editors, *High-Level Connectionist Models*, vol. 1 of *Advances in Connectionist and Neural Computation Theory*, Barnden, J. A., series editor, 17–31. Norwood, NJ: Ablex.

Zelle, J. M., and Mooney, R. J. (1993). Learning semantic grammars with constructive inductive logic programming. In *Proceedings of the 11th National Conference on Artificial Intelligence*, 817–822. Cambridge, MA: MIT Press.

Zelle, J. M., and Mooney, R. J. (1996). Comparative results on using inductive logic programming for corpus-based parser construction. In Wermter, S., Riloff, E., and Scheler, G., editors, *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing*, 355–369. Berlin; New York: Springer.