Using a Sequential SOM to Parse Long-term Dependencies

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

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

Abstract

Simple Recurrent Networks (SRNs) have been widely used in natural language processing tasks. However, their ability to handle long-term dependencies between sentence constituents is somewhat limited. NARX networks have recently been shown to outperform SRNs by preserving past information in explicit delays from the network's prior output. However, it is unclear how the number of delays should be determined. In this study on a shift-reduce parsing task, we demonstrate that comparable performance can be derived more elegantly by using a SARDNET self-organizing map. The resulting architecture can represent arbitrarily long sequences and is cognitively more plausible.

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, 1993; Plaut and Shallice, 1992).

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). However, this promising line of research has been hampered by the SRN's inability to handle long-term dependencies, which abound in natural language tasks.

Another class of recurrent neural networks called Nonlinear AutoRegressive models with eXogenous inputs (NARX; Chen et al., 1990; Lin et al., 1996) have been proposed as an alternative to SRNs that can better deal with such long-term dependencies. In NARX networks, previous sequence constituents are explicitly represented in a predetermined number of output delays, thus reducing the effects of vanishing gradients, which is the primary source of memory degradation in recurrent networks (Bengio et al., 1994). The performance of these networks is strongly dependent on the number of output delays, and there are no guidelines on how many are needed.

This paper describes a method of extending recurrent networks such as the SRN and NARX with SARDNET (James and Miikkulainen, 1995), a self-organizing map algorithm designed to represent sequences. SARDNET permits the sequence information to remain explicit, yet generalizable in the sense that similar sequences result in similar patterns on the map. When SARDNET is coupled with SRN or NARX, the resulting networks perform better in the shift-reduce parsing task taken up in this study. Even with no recurrency and no explicit delays, the performance is almost as good. These results show that SARDNET can be used as an effective, concise, and elegant sequence memory in natural language processing tasks, and the approach should also scale up well to realistic language.

The Task: Shift-Reduce Parsing

Shift-reduce (SR) parsing is one of the simplest approaches to sentence processing that has the potential to handle a substantial subset of English (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,

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 onto the stack, or reduces the top elements of the stack into a higher-level representation, such as **the boy → NP[the,boy]** (step 3). (Phrase labels such as "NP" and "RC" are only used in this figure to make the process clear.)

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, 1996). It is also possible to train a neural network to make parsing decisions based on the current stack and the input buffer. If trained properly, the neural network can generalize well to new sentences (Simmons and Yu, 1992). Whatever correlations there exist between the word representations and the appropriate shift/reduce 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 (Recursive Auto-Associative Memory) auto-encoding network (Pollack, 1990). The SPEC architecture, however, was not a complete implementation of SR parsing; it was designed specifically for embedded relative clauses. For general parsing, the stack needs to be encoded with neural networks to make it possible to parse 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 subsymbolic parsing network must first be solved.

Parser architectures

A subsymbolic parser is a recurrent network such as SRN or NARX. 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 (SRN) or prior outputs (NARX) 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.

Recurrent neural networks can be used to implement a shift-reduce parser in the following way (figure 2: 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). 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. The parser can be implemented with various network architectures; the SRN and NARX networks are compared in this study.

SRN

In the simple recurrent network, the hidden layer is saved and fed back into the network at each step during sentence processing. The network is typically trained using the standard backpropagation algorithm (Rumelhart et al., 1986). 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. As a result, parsing with an SRN has been limited to relatively simple sentences with shallow structure.

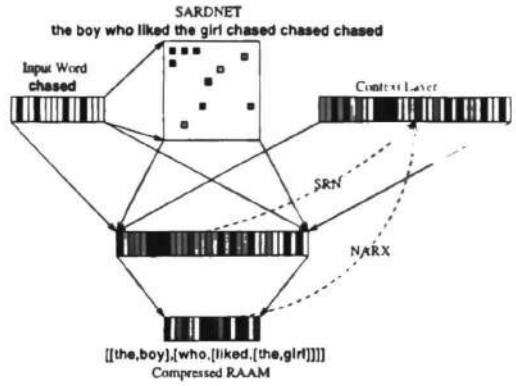

Figure 2: **The Parser Network.** This snapshot shows the network during step 11 of figure 1. The representation for the current input word, **chased**, is shown at top left. Each word is input to the SARDNET map, which builds a representation for the sequence word by word. In the SRN implementation of the parser, the previous activation of the hidden layer is copied (as indicated by the dotted line labelled SRN) to the Context assembly at each step. In the NARX implementation, a predetermined number of previous output representation compose the Context (indicated by the dotted line labelled NARX). The Context, together with the current input word and the current SARDNET pattern, is propagated to the hidden layer of the 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 is trained through the Self-Organizing Map (SOM) algorithm (Kohonen, 1997, 1990). All other connections are trained through backpropagation (for SRN) or BPTT (for NARX) (Rumelhart et al., 1986; Werbos, 1974).

NARX

Nonlinear AutoRegressive models with eXogenous inputs (NARX; Chen et al., 1990; Lin et al., 1996) have been proposed as an alternative to SRNs. They are good at dealing with long-term dependencies that typically arise in nonlinear systems such as system identification (Chen et al., 1990), time series prediction (Connor et al., 1992), and grammatical inference (Horne and Giles, 1995). NARX is a feedforward network with copies of previous outputs called delays fed back into the network during sequence processing. The network is trained via BackPropagation through Time (Rumelhart et al., 1986; Werbos, 1974), which allows "vanishing gradient" information to influence later outputs by unfolding the network in time. The performance of the network improves in an exponentially decreasing manner with the number of output delays provided.

SARDNET

The solution described in this paper is to use an explicit representation of the input sequence on a self-organizing map 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 is also not limited by length.

The SARDNET (Sequential Activation Retention and Decay Network; James and Miikkulainen, 1995) used for this purpose is based on the Self-Organizing Map neural network (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 sentence 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). Once a unit is activated, it is removed from competition and cannot represent later words in the sequence. Each unit may then represent different words depending on the context, which allows for an efficient representation of sequences, and also generalizes well to new sequences.

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

Experiments

Input Data

The data used to train and test the SRN and SARDSRN networks was generated from the phrase structure grammar in figure 3, adapted from a grammar that has become common in the literature (Elman, 1991; Miikkulainen, 1996), but limited to a maximum of 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, 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 **[who,[liked,[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, 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.

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 eight units was repeated eight times to make a redundant 64-unit representation.

Rule Schemata	$S \rightarrow NP(n)\ VP(n,m)$ $RC(n) \rightarrow$ **who** $VP(n,m)$	$VP(n,m) \rightarrow Verb(n,m)\ NP(m)$ $NP(n) \rightarrow$ **the** $Noun(n)\ RC(n)$	$NP(n) \rightarrow$ **the** $Noun(n)$ $RC(n) \rightarrow$ **whom** $NP(m)\ Verb(m,n)$
Nouns	$Noun(0) \rightarrow$ **boy**	$Noun(1) \rightarrow$ **girl**	$Noun(2) \rightarrow$ **dog**	$Noun(3) \rightarrow$ **cat**
Verbs	$Verb(0,0) \rightarrow$ **liked, saw** $Verb(0,3) \rightarrow$ **chased** $Verb(1,2) \rightarrow$ **liked** $Verb(2,1) \rightarrow$ **bit** $Verb(3,0) \rightarrow$ **saw**	$Verb(0,1) \rightarrow$ **liked, saw** $Verb(1,0) \rightarrow$ **liked, saw** $Verb(1,3) \rightarrow$ **chased** $Verb(2,2) \rightarrow$ **bit** $Verb(3,1) \rightarrow$ **saw**	$Verb(0,2) \rightarrow$ **liked** $Verb(1,1) \rightarrow$ **liked, saw** $Verb(2,0) \rightarrow$ **bit** $Verb(2,3) \rightarrow$ **bit, chased** $Verb(3,3) \rightarrow$ **chased**

Figure 3: **Grammar.** This phrase structure grammar generates sentences with subject- and object-extracted relative clauses. The rule schemata with noun and verb restrictions ensure semantic agreement between subject and object depending on the verb in the clause. Lexicon items are given in bold face.

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 (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.

System Parameters and Training

SARDNET maps were added to the SRN and NARX networks to yield SARDSRN and SARDNARX parsing architectures. Additionally, a SARDNET map was added to a simple feedforward network (FFN). This (SARDFFN) network provides a baseline for evaluating the map itself in the parsing task. The performances of all the architectures was compared in the shift-reduce parsing task. Additionally, for the NARX and SARDNARX networks, delays of 0, 3, 6, 9, 12, and 15 prior inputs (covering almost the entire sentence) were constructed. The size of the hidden layer for each network was determined so that the total number of weights was as close to 64,000 as the topology would permit (figure 6).

Four data sets of 20%, 40%, 60%, and 80% of the 436 sentences generated by the grammar were randomly selected and each parser was trained on each dataset four times. Training on all 256 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. All networks were trained with a learning rate of 0.1, and the maps had a decay rate of 0.9. A map of 100 units was pretrained with a learning rate of 0.6, and then used for all of the SARDNARX networks. A slightly larger map with 144 units was used for the SARDFFN and SARDSRN networks since these networks had otherwise much fewer weights. Training took about one day on a 400 MHz Pentium Pro workstation for each network.

Results

Epoch error, the average error per output unit during each epoch, is usually used to gauge the networks' performance in experimental studies like this. It 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 incorrect decoding of the compressed representations of the parse tree.

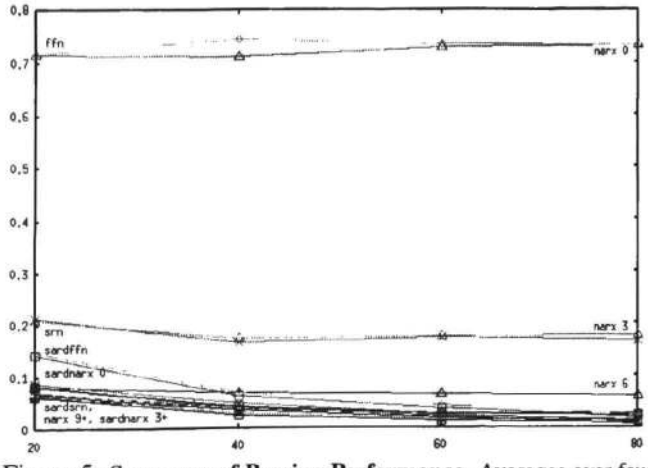

Figure 5: **Summary of Parsing Performance.** Averages over four simulations each for the fifty two networks tested using the stricter *average mismatches* per sentence measure on the test data. Most of the differences among the SARD networks and NARX networks with nine or more delays were statistically insignificant. SRN, NARX-3, SARDNARX-0, and NARX-6 differ in some of the data points as can be seen is this plot. The FFN and NARX-0 networks provide baselines of how simple feedforward networks would perform on this task (with regular BP and with BPTT), and the SRN shows how much simple recurrency helps. By comparison, the SARDFFN, SARDNARX-0, and SARDSRN graphs demonstrate that storing sequence information on SARDNET can significantly improve performance, while adding delays will improve that performance only minimally.

A more informative measure, *average mismatches*, therefore, was used in the comparisons. This measure reports the average number of leaf representations per sentence that could not be correctly identified by nearest match in Euclidean distance from the lexicon. As an example, if the target is **[who,[liked,[the,girl]]]]** (step 11 of figure 1), but the output is **[who,[saw,[the,girl]]]]**, then a mismatch would occur at the leaf labelled **saw** once the RAAM representation was decoded. Average mismatches provides a measure of the correctness of the information in the RAAM representation, and is a true measure of the utility of the network.

Most of the sentences in the training and test datasets were

network	delays	hidden layer	weights	network	delays	hidden layer	map size	weights
FFN		500	64000	SARDFFN		201	144	63888
SRN		197	64025	SARDSRN		134	144	64016
NARX	0	500	64000	SARDNARX	0	252	100	63856
NARX	3	200	64000	SARDNARX	3	137	100	63940
NARX	6	125	64000	SARDNARX	6	94	100	63928
NARX	9	91	64064	SARDNARX	9	71	100	63484
NARX	12	72	64512	SARDNARX	12	58	100	64168
NARX	15	59	64192	SARDNARX	15	48	100	63424

Figure 6: **Network Parameters.** In order to keep the network size as consistent as possible, the number of units in the hidden layers size was varied according to the size of the inputs. Because the SARD networks included a 100-unit map (144 units in the SARDFFN and SARDSRN) that was connected to both the input and hidden layers, the size of the hidden layer was proportionally made smaller.

seventeen words long. The longest long-term dependency the networks had to overcome was at step three in the parsing process where the first reduction occurred, which was part of the final compressed RAAM parse representation for the complete sentence. It was in decoding this final parse representation that even the best networks made errors. The results are summarized in figure 5. The main result is that the performance of all the SARD networks is comparable to NARX with nine or more delays, and clearly superior to SRN and NARX with zero to six delays. These results demonstrate that adding SARDNET to a recurrent network results in very robust performance without the expense of restricting the network to a prespecified number of delays. However, if the domain is well-behaved enough that this number can be determined, adding such delays will improve performance somewhat.

The SARDFFN and SARDNARX results are slightly weaker with the 20% dataset. On closer inspection it turned out that the map was not smooth enough to allow as good generalization as in the larger datasets, where there was sufficient data to overcome the map irregularities. It is also interesting to note that adding even a single delay completely eliminated this problem, bringing the performance in line with the others. We plan to improve generalization on the map further in future work.

Discussion

These results demonstrate a practicable solution to the memory degradation problem of recurrent networks. When prior constituents are explicitly represented at the input, the recurrent network does not have to maintain specific information about the sequence, 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 recurrent networks augmented with SARDNET or with delays.

These results also show that networks with SARDNET can perform as well as NARX networks with many delays. Why is this a useful result? The point is that it will always be unclear how many delays are needed in a NARX network, whereas SARDNET can accommodate sequences of indefinite length (limited only by the number of nodes in the network). This relieves the designer from having to specify, by trial and error, the appropriate number of delays. It should also lead to more graceful degradation with unexpectedly long sequences, and therefore would allow the system to scale up better and exhibit more plausible cognitive behavior.

The operation of the recurrent networks 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.

The SARDNET 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 or explicit delays (and how many) is an open question, but we believe the SARDNET representation suggests an elegant way to capture a lot of the resulting behavior. SARDNET is a plausible cognitive approach, and useful for building powerful subsymbolic language understanding systems. SARDNET is also in line with the general neurological evidence for topographical maps in the brain.

Conclusion

We have shown how explicit representation of constituents on a self-organizing map allows recurrent networks to process sequences more effectively. We demonstrated that neural networks equipped with SARDNET sequence memory achieve comparable performance as NARX networks with many delays in a nontrivial shift-reduce parsing task. SARDNET, however, is more elegant and cognitively plausible in that it does not impose limits on the length of the sequences it can process. In future work, we will examine how exactly the networks are using the map information to improve the generalization ability of SARDNET, as well as extending the method to other recurrent neural network architectures.

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 In-*

ternational Conference on Neural Networks (San Diego, CA), volume II, pages 335–341, Piscataway, NJ. IEEE.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

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

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

Chen, S., Billings, S., and Grant, P. (1990). Non-linear system identification using neural networks. In *International Journal of Control*, pages 1191–1214.

Connor, J., Atlas, L., and Martin, D. (1992). Recurrent networks and narma modeling. *Advances in Neural Information Processing Systems*, 4:301–308.

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*.

Horne, B. and Giles, C. (1995). An experimental comparison of recurrent neural networks. *Advances in Neural Information Processing Systems*, 7:697–704.

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*, pages 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*. Springer, Berlin; New York, second edition.

Lin, T., Horne, B. G., and Giles, C. L. (1996). Learning long-term dependencies in narx recurrent neural networks. *IEEE Transactions on Neural Networks*, 7(6):1329–1338.

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, Volume 2: Psychological and Biological Models*, pages 272–325. MIT Press, Cambridge, MA.

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

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.

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.

Plaut, D. C. and Shallice, T. (1992). Perseverative and semantic influences on visual object naming errors in optic aphasia: A connectionist account. Technical Report PDP.CNS.92.1, Parallel Distributed Processing and Cognitive Neuroscience, Department of Psychology, Carnegie Mellon University, Pittsburgh, PA.

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

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing, Volume 1: Foundations*, pages 318–362. MIT Press, Cambridge, MA.

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*, pages 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, ICSI, Berkeley, CA.

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

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

Werbos, P. J. (1974). *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Department of Applied Mathematics, Harvard University, Cambridge, MA.

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*, pages 355–369. Springer, Berlin; New York.