Parsing Embedded Clauses with Distributed Neural Networks

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

Dennis Bijwaard
Department of Computer Science
University of Twente
7500 AE Enschede, The Netherlands
bijwaard@cs.utwente.nl

Abstract

A distributed neural network model called SPEC for processing sentences with recursive relative clauses is described. The model is based on separating the tasks of segmenting the input word sequence into clauses, forming the case-role representations, and keeping track of the recursive embeddings into different modules. The system needs to be trained only with the basic sentence constructs, and it generalizes not only to new instances of familiar relative clause structures, but to novel structures as well. SPEC exhibits plausible memory degradation as the depth of the center embeddings increases, its memory is primed by earlier constituents, and its performance is aided by semantic constraints between the constituents. The ability to process structure is largely due to a central executive network that monitors and controls the execution of the entire system. This way, in contrast to earlier subsymbolic systems, parsing is modeled as a controlled high-level process rather than one based on automatic reflex responses.

Introduction

Reading an input sentence into an internal representation is a most fundamental task in natural language processing. In the distributed (i.e. subsymbolic) neural network approach, it usually involves mapping a sequence of word representations into a shallow semantic interpretation, such as the case-role assignment of the constituents. This approach offers several promises: it is possible to combine syntactic, semantic, and thematic constraints in the interpretation, generate expectations automatically, generalize to new inputs, and process noisy sentences robustly (Elman 1990, 1991; McClelland & Kawamoto 1986; Miikkulainen 1993; St. John & McClelland 1990). To a limited extent, it is even possible to train such networks to process sentences with complex grammatical structure, such as embedded relative clauses (Berg 1992; Jain 1991; Miikkulainen 1990; Sharkey & Sharkey 1992; Stolcke 1990).

However, it has been very difficult to build subsymbolic systems that would generalize to new sentence structures. A network can be trained to form a case-role representation of each clause in a sentence like `The girl, who liked the dog, saw the boy`[1], and it will generalize to different versions of the same structure, such as `The dog, who bit the girl, chased the cat` (Miikkulainen 1990). However, such a network cannot parse sentences with novel combinations of relative clauses, such as `The girl, who liked the dog, saw the boy, who chased the cat`. The problem is that distributed neural networks are pattern transformers, and they generalize by interpolating between patterns on which they were trained. They cannot make inferences by dynamically combining processing knowledge that was previously associated to different contexts, such as processing a relative clause at a new place in an otherwise familiar sentence structure. This lack of generalization is a serious problem, given how effortlessly people can understand sentences they have never seen before.

This paper describes SPEC (Subsymbolic Parser for Embedded Clauses), a subsymbolic sentence parsing model that can generalize to new relative clause structures. The basic idea is to separate the tasks of segmenting the input word sequence into clauses, forming the case-role representations, and keeping track of the recursive embeddings into different networks. Each network is trained with only the most basic relative clause constructs, and the combined system is able to generalize to novel sentences with remarkably complex structure. Importantly, SPEC is not a neural network reimplementation of a symbol processor. It is a self-contained, purely distributed neural network system, and exhibits the usual properties of such systems. For example, unlike symbolic parsers, the network exhibits plausible memory degradation as the depth of the center embeddings increases, its memory is primed by the earlier constituents in the sentence, and its performance is aided by semantic constraints between the constituents.

The SPEC Architecture

SPEC receives a sequence of word representations as its input, and for each clause in the sentence, forms an output representation indicating the assignment of

[1] In all examples in this paper, commas are used to indicate clause boundaries for clarity.

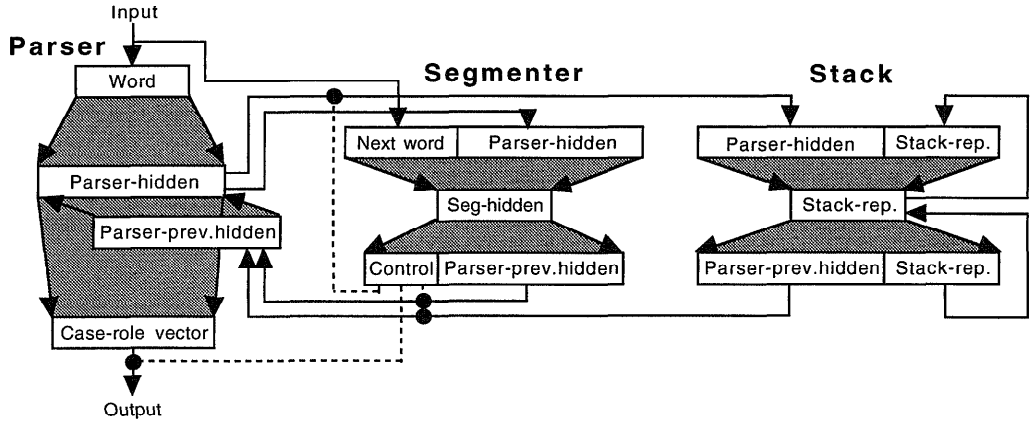

Figure 1: **The SPEC sentence processing architecture.** The system consists of the Parser (a simple recurrent network), the Stack (a RAAM network), and the Segmenter (a feedforward network). The gray areas indicate propagation through weights, the solid lines stand for pattern transport, and the dashed lines represent control outputs (with gates).

words into case roles. The case-role representations are read off the system and placed in a short-term memory (currently outside SPEC) as soon as they are complete. SPEC consists of three main components: the Parser, the Segmenter, and the Stack (figure 1). Below, each component is described in detail.

The Parser

The Parser performs the actual transformation of the word sequence into the case-role representations, and like many other subsymbolic parsers, it is based on Elman's (1990) simple recurrent network architecture (SRN; figure 2). The pattern in the hidden layer is copied to the previous-hidden-layer assembly and serves as input to the hidden layer during the next step in the sequence, thus implementing a sequence memory. The network is trained with examples of input/output sequences, adjusting all forward weights according to the backpropagation algorithm (Rumelhart, Hinton, & Williams 1986).

Words are represented distributively as vectors of gray-scale values between 0 and 1. The component values are initially assigned randomly and modified during learning by the FGREP method (Miikkulainen & Dyer 1991; Miikkulainen 1993). FGREP is a convenient way to form distributed representations for input/output items, but SPEC is not dependent on FGREP. The word representations could have been obtained through semantic feature encoding (McClelland & Kawamoto 1986) as well, or even assigned randomly.

The case-role assignment is represented at the output of the Parser as a case-role vector (CRV), that is, a concatenation of those three word representation vectors that fill the roles of agent, act, and patient in the sentence[2] (figure 2). For example, the word sequence `the girl saw the boy` receives the case-role assignment agent=`girl`, act=`saw`, patient=`boy`, which is represented as the vector `|girl saw boy|` at the output of the Parser network. When the sentence consists of multiple clauses, the relative pronouns are replaced by their referents: `The girl, who liked the dog, saw the boy` parses into two CRVs: `|girl liked dog|` and `|girl saw boy|`.

The Parser receives a continuous sequence of input word representations as its input, and its target pattern changes at each clause boundary. For example, in reading `The girl, who liked the dog, saw the boy`, the target pattern representing `|girl saw boy|` is maintained during the first two words, then switched to `|girl liked dog|` during reading the embedded clause, and then back to `|girl saw boy|` for the rest of the sentence. The CRV for the embedded clause is read off the network after `dog` has been input, and the CRV for the main clause after the entire sentence has been read.

When trained this way, the network is not limited to a fixed number of clauses by its output representation. Also, it does not have to maintain information about the entire past input sequence in its memory, making it possible in principle to generalize to new clause structures. Unfortunately, after a center-embedding has been processed, it is difficult for the network to remember earlier constituents. This is why a Stack network is needed in SPEC.

The Stack

The hidden layer of a simple recurrent network forms a compressed description of the sequence so far. The Stack has the task of storing this representation at each center embedding, and restoring it upon return from the embedding. For example, in parsing `The girl, who liked the dog, saw the boy`, the hidden-layer

[2]The representation was limited to three roles for simplicity.

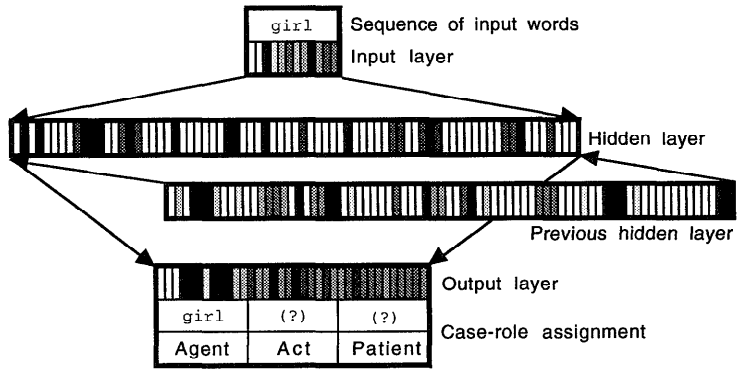

Figure 2: **The Parser network.** The figure depicts a snapshot of the network after it has read the first two words The and girl. The activity patterns in the input and output assemblies consist of word representations. The input layer holds the representation for the last word, girl, and the activity pattern at the output represents the (currently incomplete) case-role assignment of the clause.

representation is pushed onto the stack after The girl, and popped back to the Parser's previous-hidden-layer assembly after who liked the dog. In effect, the SRN can then parse the top-level clause as if the center embedding had not been there at all.

The Stack is implemented as a Recursive Auto-Associative Memory (RAAM; Pollack 1990; figure 3). RAAM is a three-layer backpropagation network trained to perform an identity mapping from input to output. As a side effect, its hidden layer learns to form compressed representations of the network's input/output patterns. These representations can be recursively used as constituents in other input patterns, and a potentially infinite hierarchical data structure, such as a stack, can this way be compressed into a fixed-size representation.

The input/output of the Stack consists of the stack's top element and the compressed representation for the rest of the stack. Initially the stack is empty, which is represented by setting all units in the "Stack" assembly to 0.5 (figure 3). The first element, such as the hidden-layer pattern of the Parser network after reading The girl, is loaded into the "Push" assembly, and the activity is propagated to the hidden layer. The hidden-layer pattern is then loaded into the "Stack" assembly at the input, and the Stack network is ready for another push operation.

When the Parser returns from the center embedding, the stored pattern needs to be popped from the stack. The current stack representation is loaded into the hidden layer, and the activity is propagated to the output layer. At the output, the "Pop" assembly contains the stored Parser-hidden-layer pattern, which is then loaded into the previous-hidden-layer assembly of the Parser network (figure 1). The "Stack" assembly contains the compressed representation for the rest of the stack, and it is loaded to the hidden layer of the Stack network, which is then ready for another pop operation.

The Segmenter

The Parser+Stack architecture alone is not quite sufficient for generalization into novel relative clause structures. For example, when trained with only examples of center embeddings (such as the above) and tail embeddings (like The girl saw the boy, who chased the cat), the architecture generalizes well to new sentences such as The girl, who liked the dog, saw the boy, who chased the cat. However, the system still fails to generalize to sentences like The girl saw the boy, who the dog, who chased the cat, bit. Even though the Stack takes care of restoring the earlier state of the parse, the Parser has to learn all the different transitions into relative clauses. If it has encountered center embeddings only at the beginning of the sentence, it cannot generalize to a center embedding that occurs after an entire full clause has already been read.

The solution is to train an additional network, the Segmenter, to divide the input sequence into clauses. The segmenter receives the current hidden-layer pattern as its input, together with the representation for the next input word, and it is trained to produce a modified hidden-layer pattern as its output (figure 4). The output is then loaded into the previous-hidden-layer assembly of the Parser. In the middle of reading a clause, the Segmenter passes the hidden-layer pattern through without modification. However, if the next word is a relative pronoun, the segmenter modifies the pattern so that only the relevant information remains. In the above example, after boy has been read and who is next to come, the Segmenter generates a pattern similar to that of the Parser's hidden layer after only The boy in the beginning of the sentence has been input.

In other words, the Segmenter (1) detects transitions to relative clauses, and (2) changes the sequence memory so that the Parser only has to deal with one type of clause boundary. This way, the Parser's task be-

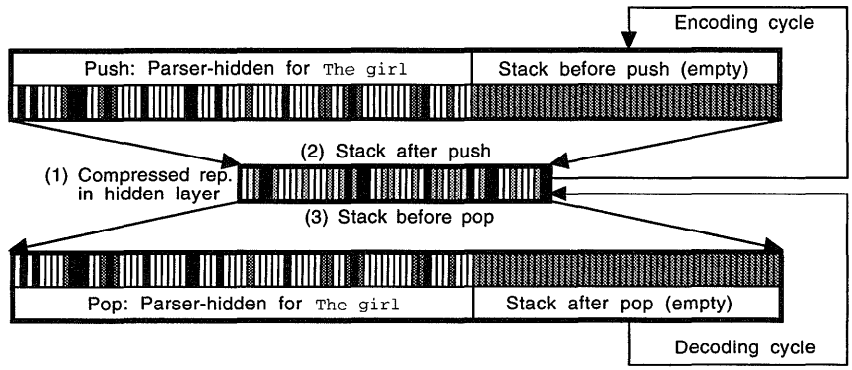

Figure 3: **The Stack network.** This figure simultaneously illustrates three situations that occur at different times during the training and the performance of the Stack: (1) A training situation where the network learns to autoassociate an input pattern with itself, forming a compressed representation at the hidden layer; (2) A push operation, where a representation in the "Push" assembly is combined with the empty-stack representation (in the "Stack" assembly) to form a compressed representation for the new stack in the hidden layer; (3) A pop operation, where the current stack representation in the hidden layer generates an output pattern with the top element of the stack in the "Pop" assembly and the representation for the remaining stack (currently empty) in the "Stack" assembly.

comes sufficiently simple so that the entire system can generalize to new structures.

The Segmenter plays a central role in the architecture, and it is very natural to give it a complete control over the entire parsing process. Control is implemented through three additional units at the Segmenter's output (figure 4). The units "Push" and "Pop" control the stack operations, and the unit "Output" indicates when the Parser output is complete and should be read off the system. The control implementation in SPEC emphasizes an important point: although much of the structure in the parsing task is programmed into the system architecture, SPEC is still a self-contained distributed neural network. In many modular neural network architectures control is due to a hidden symbolic supervisor. SPEC demonstrates that such external control mechanisms are not necessary: even a rather complex subsymbolic architecture can take care of its own control and operate independently of its environment.

Experiments

The training and testing corpus was generated from a simple phrase structure grammar (table 1). Each clause consisted of three constituents: the agent, the verb and the patient. A relative who-clause could be attached to the agent or to the patient of the parent clause, and who could fill the role of either the agent or the patient in the relative clause. In addition to who, the and "." (full stop, the end-of-sentence marker that had its own distributed representation in the system just like a word), the vocabulary consisted of the verbs chased, liked, saw and bit, and the nouns boy, girl, dog and cat. Certain semantic restrictions were imposed on the sentences. A verb could only have certain nouns as its agent and patient, as listed in table 2. The grammar was used to generate all sentences with up to four

```
S    → NP VP "."
NP   → DET N | DET N RC
VP   → V NP
RC   → who VP | who NP V
N    → boy | girl | dog | cat
V    → chased | liked | saw | bit
DET  → the
```

Table 1: **The sentence grammar.**

Verb	Case-role	Possible fillers
chased	Agent:	boy,girl,dog,cat
	Patient:	cat
liked	Agent:	boy,girl
	Patient:	boy,girl,dog
saw	Agent:	boy,girl,cat
	Patient:	boy,girl
bit	Agent:	dog
	Patient:	boy,girl,dog,cat

Table 2: **Semantic restrictions.**

clauses, and those that did not match the semantic restrictions were discarded. The final corpus consisted of 49 different sentence structures, with a total of 98,100 different sentences.

The SPEC architecture divides the sentence parsing task into three subtasks. Each component needs to learn only the basic constructs in its task, and the combined architecture forces generalization into novel combinations of these constructs. Therefore, it is enough to train SPEC with only two sentence structures: (1) the two-level tail embedding (such as The girl saw the boy, who chased the cat, who the dog bit) and the two-level center-embedding (e.g. the girl, who the dog, who chased the cat, bit, saw the boy). The training set consisted of 100 randomly-selected sen-

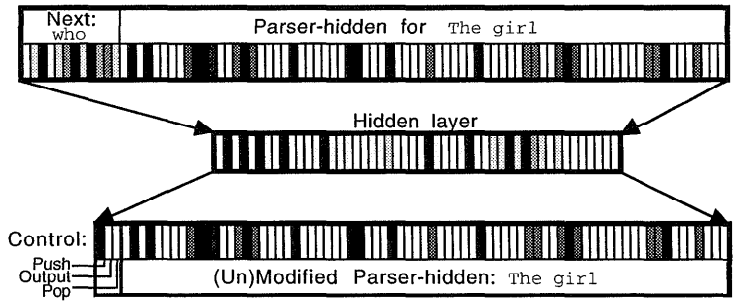

Figure 4: **The Segmenter network.** The Segmenter receives the Parser's hidden-layer pattern as its input together with the next input word, which in this case is `who`. The control outputs are 1, 0, 0, indicating that the Parser's hidden-layer representation should be pushed onto the Stack, the current case-role representation is incomplete and should not be passed on to the output of the system, and the stack should not be popped at this point. In this case, the Segmenter output is identical to its input, because `the girl` is the smallest context that the Parser needs to know when entering a center embedding.

tences of each type. In addition, the Stack was trained to encode and decode up to three levels of pushes and pops.

The word representations consisted of 12 units. Parser's hidden layer was 75 units wide, Segmenter's 50 units, and Stack's 50 units. All networks were trained with on-line backpropagation with 0.1 learning rate and without momentum. Both the Parser and the Segmenter developed word representations at their input layers (with a learning rate of 0.001). The networks were trained separately (i.e. without propagation between modules) and simultaneously, sharing the same gradually-developing word and parser-hidden-layer representations. The convergence was very strong. After 400 epochs, the average error per output unit was 0.018 for the Parser, 0.008 for the Segmenter (0.002 for the control outputs), and 0.003 for the Stack, while an error level of 0.020 usually results in acceptable performance in similar assembly-based systems (Miikkulainen 1993). The training took approximately three hours on an IBM RS6000 workstation. The final representations reflected the word categories very well.

SPEC's performance was then tested on the entire corpus of 98,100 sentences. The patterns in the Parser's output assemblies were labeled according to the nearest representation in the lexicon. The control output was taken to be correct if those control units that should have been active at 1 had an activation level greater than 0.7, and those that should have been 0 had activation less than 0.3. Measured this way, the performance was excellent: SPEC did not make a single mistake in the entire corpus, neither in the output words or in control. The average unit error was 0.034 for the Parser, 0.009 for the Segmenter (0.003 for control), and 0.005 for the Stack. There was very little variation between sentences and words within each sentence, indicating that the system was operating within a safe margin.

The main result, therefore, is that the SPEC architecture successfully generalizes not only to new instances of the familiar sentence structures, but to new structures as well, which the earlier subsymbolic sentence processing architectures could not do. However, SPEC is not a mere reimplementation of a symbol processor. As SPEC's Stack becomes increasingly loaded, its output becomes less and less accurate; symbolic systems do not have any such inherent memory degradation. An important question is, does SPEC's performance degrade in a cognitively plausible manner, that is, does the system have similar difficulties in processing recursive structures as people do?

To elicit enough errors from SPEC to analyze its limitations, the Stack's performance was degraded by adding 30% noise in its propagation. Such an experiment can be claimed to simulate overload, stress, cognitive impairment, or lack of concentration situations. The system turned out to be remarkably robust against noise. The average Parser error rose to 0.058, but the system still got 94% of its output words right, with very few errors in control. As expected, most of the errors occurred as a direct result of popping back from center embeddings with an inaccurate previous-hidden-layer representation. For example, in parsing `The girl, who the dog, who the boy, who chased the cat, liked, bit, saw the boy`, SPEC had trouble remembering the agents of `liked`, `bit` and `saw`, and patients of `liked` and `bit`. The performance depends on the level of the embedding in an interesting manner. It is harder for the network to remember the earlier constituents of shallower clauses than those of deeper clauses. For example, SPEC could usually connect `boy` with `liked` (in 80% of the cases), but it was harder for it to remember that it was the `dog` who `bit` (58%) and even harder that the `girl` who `saw` (38%) in the above example.

Such behavior seems plausible in terms of human performance. Sentences with deep center embeddings are harder for people to remember than shallow ones (Foss & Cairns 1970; Miller & Isard 1964). It is easier

to remember a constituent that occurred just recently in the sentence than one that occurred several embeddings ago. Interestingly, even though SPEC was especially designed to overcome such memory effects in the Parser's sequence memory, the same effect is generated by the Stack architecture. The latest embedding has noise added to it only once, whereas the earlier elements in the stack have been degraded multiple times. Therefore, the accuracy is a function of the number of pop operations instead of a function of the absolute level of the embedding.

When the SPEC output is analyzed word by word, several other interesting effects are revealed. Virtually in every case where SPEC made an error in popping an earlier agent or patient from the stack it confused it with another noun (54,556 times out of 54,603; random choice would yield 13650). In other words, SPEC performs plausible role bindings: even if the exact agent or patient is obscured in the memory, it "knows" that it has to be a noun. Moreover, SPEC does not generate the noun at random. Out of all nouns it output incorrectly, 75% had occurred earlier in the sentence, whereas a random choice would give only 54%. It seems that traces for the earlier nouns are discernible in the previous-hidden-layer pattern, and consequently, they are slightly favored at the output. Such priming effect is rather surprising, but it is very plausible in terms of human performance.

The semantic constraints (table 2) also have a marked effect on the performance. If the agent or patient that needs to be popped from the stack is strongly correlated with the verb, it is easier for the network to remember it correctly. The effect depends on the strength of the semantic coupling. For example, `girl` is easier to remember in `The girl, who the dog bit, liked the boy,` than in `The girl, who the dog bit, saw the boy,` which is in turn easier than `The girl, who the dog bit, chased the cat.` The reason is that there are only two possible agents for `liked`, whereas there are three for `saw` and four for `chased`. While SPEC gets 95% of the unique agents right, it gets 76% of those with two alternatives, 69% of those with three, and only 67% of those with four.

A similar effect has been observed in human processing of relative clause structures. Half the subjects in Stolz's (1967) study could not decode complex center embeddings without semantic constraints. Huang (1983) showed that young children understand embedded clauses better when the constituents are semantically strongly coupled, and Caramazza & Zurif (1976) observed similar behavior in aphasics. This effect is often attributed to limited capability for processing syntax. The SPEC experiments indicate that it could be at least partly due to impaired memory as well. When the memory representation is impaired with noise, the Parser has to clean it up. In propagation through the Parser's weights, noise that does not coincide with the known alternatives cancels out. Apparently, when the verb is strongly correlated with some of the alternatives, more of the noise appears coincidental and is filtered out.

Discussion

Several observations indicate that the SPEC approach to subsymbolic parsing should scale up well. First, as long as SPEC can be trained with the basic constructs, it will generalize to a very large set of new combinations of these constructs. Combinatorial training (St. John 1992) of structure is not necessary. In other words, SPEC is capable of *dynamic inferencing*, previously postulated as very difficult for subsymbolic systems to achieve (Touretzky 1991). Second, like most subsymbolic systems, SPEC does not need to be trained with a complete set of all combinations of constituents for the basic constructs; a representative sample, like the 200 out of 1088 possible training sentences above, is enough. Third, with the FGREP mechanism it is possible to automatically form meaningful distributed representations for a large number of words, even to acquire them incrementally (Miikkulainen & Dyer 1991; Miikkulainen 1993), and the network will know how to process them in new situations. Fourth, SPEC is quite insensitive to configuration and simulation parameters, suggesting that the approach is very strong, and there should be plenty of room for adapting it to more challenging experiments. The most immediate direction for future work is to apply the SPEC architecture to a wider variety of grammatical constructs and to larger vocabularies.

The Segmenter is perhaps the most significant new feature of the SPEC architecture. It can be seen as a first step toward implementing high-level control in the connectionist framework (see also Jacobs, Jordan, & Barto 1991; Jain 1991; Schneider & Detweiler 1987; Sumida 1991). The Segmenter monitors the input sequence and the state of the parsing network, and issues I/O control signals for the Stack memory and the Parser itself at appropriate times. The Segmenter has a high-level view of the parsing process, and uses it to assign simpler tasks to the other modules. In that sense, the Segmenter implements a strategy for parsing sentences with relative clauses. Such control networks could play a major role in future subsymbolic models of natural language processing and high-level reasoning.

Conclusion

SPEC is largely motivated by the desire to build a system that (1) would be able to process nontrivial input like symbolic systems, and (2) would make use of the unique properties of distributed neural networks such as learning from examples, spontaneous generalization, robustness, context sensitivity, and integrating statistical evidence. Although SPEC does not address several important issues in connectionist natural language processing (such as processing exceptions and representing flexible structure), it does indicate that learning and

applying grammatical structure for parsing is possible with pure distributed networks.

However, even more than an AI system aiming at best possible performance, SPEC is an implementation of a particular Cognitive Science philosophy. The architecture is decidedly not a reimplementation of a symbol processor, or even a hybrid system consisting of subsymbolic components in an otherwise symbolic framework. SPEC aims to model biological information processing at a specific, uniform level of abstraction, namely that of distributed representation on modular networks. SPEC should be evaluated according to how well its behavior matches that produced by the brain at the cognitive level. The memory degradation experiments indicate that SPEC is probably on the right track, and the success of the high-level controller network in generating high-level behavior opens exciting possibilities for future work.

References

Berg, G. 1992. A connectionist parser with recursive sentence structure and lexical disambiguation. In *Proceedings of the 10th National Conference on Artificial Intelligence*, 32–37. Cambridge, MA: MIT Press.

Caramazza, A., and Zurif, E. B. 1976. Dissociation of algorithmic and heuristic processes in language comprehension: Evidence from aphasia. *Brain and Language* 3:572–582.

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.

Foss, D. J., and Cairns, H. S. 1970. Some effects of memory limitation upon sentence comprehension and recall. *Journal of Verbal Learning and Verbal Behavior* 9:541–547.

Huang, M. S. 1983. A developmental study of children's comprehension of embedded sentences with and without semantic constraints. *Journal of Psychology* 114:51–56.

Jacobs, R. A.; Jordan, M. I.; and Barto, A. G. 1991. Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. *Cognitive Science* 15:219–250.

Jain, A. N. 1991. Parsing complex sentences with structured connectionist networks. *Neural Computation* 3:110–120.

McClelland, J. L., and Kawamoto, A. H. 1986. Mechanisms of sentence processing: Assigning roles to constituents. In McClelland, J. L., and Rumelhart, D. E., eds., *Parallel Distributed Processing*. Cambridge, MA: MIT Press. 272–325.

Miikkulainen, R. 1990. A PDP architecture for processing sentences with relative clauses. In Karlgren, H., ed., *Proceedings of the 13th International Conference on Computational Linguistics*, 201–206. Helsinki, Finland: Yliopistopaino.

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

Miikkulainen, R., and Dyer, M. G. 1991. Natural language processing with modular neural networks and distributed lexicon. *Cognitive Science* 15:343–399.

Miller, G. A., and Isard, S. 1964. Free recall of self-embedded English sentences. *Information and Control* 7:292–303.

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., eds., *Parallel Distributed Processing*. Cambridge, MA: MIT Press. 318–362.

Schneider, W., and Detweiler, M. 1987. A connectionist/control architecture for working memory. In Bower, G. H., ed., *The Psychology of Learning and Motivation*, volume 21. New York: Academic Press. 53–119.

Sharkey, N. E., and Sharkey, A. J. C. 1992. A modular design for connectionist parsing. In Drossaers, M. F. J., and Nijholt, A., eds., *Twente Workshop on Language Technology 3*, 87 96. Department of Computer Science, University of Twente, the Netherlands.

St. John, M. F. 1992. The story gestalt: A model of knowledge-intensive processes in text comprehension. *Cognitive Science* 16:271–306.

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.

Stolz, W. S. 1967. A study of the ability to decode grammatically novel sentences. *Journal of Verbal Learning and Verbal Behavior* 6:867–873.

Sumida, R. A. 1991. Dynamic inferencing in parallel distributed semantic networks. In *Proceedings of the 13th Annual Conference of the Cognitive Science Society*, 913–917. Hillsdale, NJ: Erlbaum.

Touretzky, D. S. 1991. Connectionism and compositional semantics. In Barnden, J. A., and Pollack, J. B., eds., *High-Level Connectionist Models*. Norwood, NJ: Ablex. 17–31.