Subsymbolic Case-Role Analysis of Sentences with Embedded Clauses *†

Risto Miikkulainen
The University of Texas at Austin

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.

1 Introduction

Reading an input sentence into an internal representation is a most fundamental task in natural language processing. Depending on the field of study and the goals involved, it has several alternative formulations. In Artificial Intelligence, parsing a sentence usually means mapping a sequence of word representations into a shallow semantic interpretation, such as the case-role assignment of the constituents. The subsymbolic (i.e. distributed neural network) approach to sentence parsing 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, 1991a; McClelland and Kawamoto 1986; Miikkulainen 1993; St. John and 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 and Sharkey 1992; Stolcke 1990; Weckerly and Elman 1992).

However, it has been very difficult to build subsymbolic systems that would generalize to new sentence structures. Current network architectures 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 they will be able to generalize to different versions of the same structure, such as `The dog, who bit the`

---

* *Cognitive Science*, in press.

† Correspondence and requests for reprints should be sent to Risto Miikkulainen, Department of Computer Sciences, The University of Texas at Austin, Austin, TX 787512-1188; `risto@cs.utexas.edu`.

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

`girl, chased the cat` (Miikkulainen 1990). However, such networks 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 the current distributed neural network architectures function simply as 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 modules. Each module 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.

A significant new aspect of SPEC as a cognitive model is that it controls its own execution. One of the modules (the Segmenter) monitors the state of the parse and the input word sequence, and issues control signals to the other networks in the system. This network is responsible for abstracting the "idea" of a relative clause from the raw training examples and enforcing generalization to novel clause structures. Such high-level control networks could play a major role in future subsymbolic cognitive models. Controlled models are not limited to straightforward pattern transformation and reflex behavior like the standard subsymbolic systems; they can potentially account for higher-level controlled cognitive processes as well.

2 Overview of Subsymbolic Sentence Processing

Sentence processing has been an active area of connectionist research for about a decade. Subsymbolic models have been developed to address a variety of issues such as semantic interpretation, learning syntax and semantics, prepositional phrase attachment, anaphora resolution, active-passive transformation, and translation (Allen 1987, 1989; Chalmers 1990; Chrisman 1992; Cosic and Munro 1988; Lee et al. 1990; Munro et al. 1991; Touretzky 1991).

A good amount of work has been done showing that networks can capture grammatical structure. For example, Servan-Schreiber et al. (1989, 1991) showed how Simple Recurrent Networks (SRNs; Elman 1990) can learn a finite state grammar. In an SRN, 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 et al. 1986b). Servan-Schreiber et al. trained an SRN with sample strings from a particular grammar, and it learned to indicate the possible next elements in the sequence. For example, given a sequence of distributed representations for elements B, T, X, X, V, and V, the network turns on two units representing X and S at its localist output layer, indicating that in this grammar, the string can continue with either X or S.

Elman (1991a, 1991b) used the same network architecture to predict a context-free language with embedded clauses. The network could not learn the language completely, but its performance was remarkably similar to human performance. It learned better when it was trained incrementally, first with simple sentences and gradually including more and more complex examples. The network could maintain contingencies over embeddings if the number of intervening elements was small. However, deep center embeddings were difficult for the network, as they are for humans. Weckerly and Elman (1992) further showed that center embeddings were harder for this network than right-branching structures, and that processing was aided by semantic constraints between the lexical items. Such behavior matches human performance very well.

The above architectures demonstrated that distributed networks build meaningful internal representations when exposed to examples of strings in a language. They did not address how such capabilities could be put to use in parsing and understanding language. McClelland and Kawamoto (1986) identified the sentence case-role assigment as a good approach. Case-role representation is a common artificial intelligence technique for describing the shallow semantic meaning of a sentence. The idea is loosely based on the theory of thematic case roles (Fillmore 1968; Cook 1989). Each act is described by the main verb and a set of semantic cases such as agent, patient, instrument, location, and recipient. The task is to decide which constituents fill these roles in the sentence. The approach is particularly well-suited for neural networks because the cases can be conveniently represented as assemblies of units that hold distributed representations, and the parsing task becomes that of mapping between distributed representation patterns. McClelland and Kawamoto showed that given the syntactic role assignment of the sentence as the input, the network could assign the correct case roles for each constituent. The network also automatically performed semantic enrichment on the word representations (which were hand-coded concatenations of binary semantic features), and disambiguated between the different senses of ambiguous words.

Miikkulainen and Dyer (1989, 1991) showed that essentially the same task can be performed from sequential word-by-word input by a simple recurrent network, and, through a technique called FGREP (Forming Global Representations with Extended backPropagation), meaningful distributed representations for the words can be automatically developed at the same time. In FGREP, the component values are assigned initially randomly within [0, 1] and modified by backpropagation as part of learning the task. The final representations reflect how the words are used in the examples, and in that sense, represent word meanings. Systems with FGREP representations generally have a strong representation of context, which results in good generalization properties, robustness against noise and damage, and automatic “filling in” of missing information. The FGREP representations can be augmented with ID information, which allows the system to process a large vocabulary even after learning only a small number of distinct meanings. In this ID+content approach, representations for e.g. `John`, `Bill`, and `Mary` are created from the FGREP representation of `human` by concatenating unique ID patterns in front of it. All these words have the same meaning for the system, and it knows how to process them even if it has never seen them before (Miikkulainen and Dyer 1991; Miikkulainen 1993).

St. John and McClelland (1989, 1990) further explored the subsymbolic approach to sentence interpretation in their Sentence Gestalt model. They aimed at explaining how syntactic, semantic, and thematic constraints are combined in sentence comprehension, and how this knowledge can be coded into the network by training it with queries. The gestalt is a hidden-layer representation of the whole sentence, built gradually from a sequence of input words by a simple recurrent network. The second part of the system (a three-layer backpropagation network) is trained to answer questions about the sentence gestalt, and in the process, useful thematic knowledge can be injected into the system.

The above three parsing architectures each built a semantic interpretation of the sentence, but they could not handle grammatically very complex sentences. Several extensions and some completely new architectures that could do that have been proposed. For example, the CLAUSES system (Miikkulainen 1990) was an extension of the SRN+FGREP case-role assignment architecture into sentences with multiple clauses. CLAUSES read clause fragments one at a time, brought together the separated constituents, and concatenated the case-role representations into a comprehensive sentence representation in its output layer. CLAUSES was limited both by the rigid output representation and also by a somewhat surprising lack of generalization into new sentence structures. On the other hand, Stolcke (1990) showed that if the output representation was made more flexible, the network was likely to forget earlier constituents. The conclusion from these two models is that straightforward applications of simple recurrent networks are unlikely to be successful in parsing and representing grammatical structure.

A number of researchers have proposed modular and more structured architectures. In Jain's (1991) Structured Incremental Parser, one module was trained to assign words into phrases, and another to assign phrases into case roles. These modules were then replicated multiple times so that the recognition of each constituent was guaranteed independent of its position in the sentence. In the final system, words were input one at a time, and the output consisted of local representations for the possible assignments of words into phrases, phrases into clauses, phrases into roles in each clause, and for the possible relationships of the clauses. A consistent activation of the output units represented the interpretation of the sentence. The system could interpret complicated sentence structures, and even ungrammatical and incomplete input. However, it did not build an explicit representation for the sentence meaning. The parse result was a description of the semantic relations of the constituents; the constituents themselves were not represented.

Berg's (1992) XERIC and Sharkey and Sharkey's (1992) parser were both based on the idea of combining a simple recurrent network with a Recursive Auto-Associative Memory (RAAM; Pollack 1990) that encodes and decodes parse trees. RAAM is a three-layer backpropagation network trained to perform an identity mapping from input to output. As a side effect, the hidden layer learns to form compressed representations of the network's input/output patterns. These representations can then be recursively used as constituents in other input patterns. A potentially infinite hierarchical data structure, such as a parse tree, can this way be compressed into a fixed-size representation. The structure can later be reconstructed by loading the compressed representations into the hidden layer and reading off the expanded representation at the output.

In Sharkey and Sharkey's model, first the RAAM network was trained to form compressed representations of syntactic parse trees. Second, an SRN network was trained to predict the next word in the sequence of words that make up the sentence. Third, a standard three-layer feedforward network was trained to map the SRN hidden-layer patterns into the RAAM parse-tree representations. During performance, a sequence of words was first read into the SRN, its final hidden layer transformed into a RAAM hidden layer, and then decoded into a parse tree with the RAAM network. Berg's XERIC worked in a similar manner, except the SRN hidden layer representations were directly decoded by the RAAM network.

All five of the above architectures can parse sentences with complex grammatical structure, and they can generalize to new sentences where constituents have been substituted with other familiar constituents. Unfortunately, generalization into new sentence structures is limited. For example, due to its rigid output representation and excessive context-sensitivity, CLAUSES could not parse `The girl, who liked the dog, saw the boy, who chased the cat`, even if it knew how to process `The girl, who liked the dog, saw the boy` and `The girl saw the boy, who`

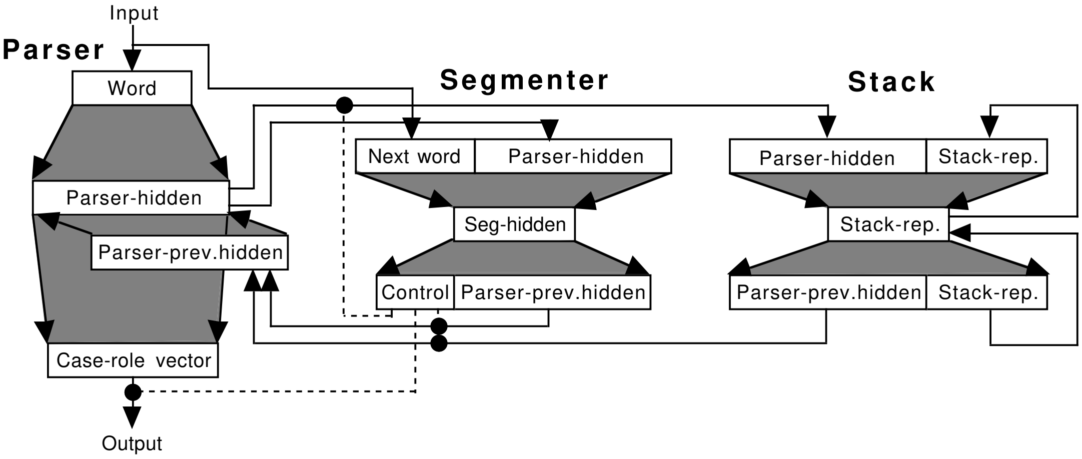

Figure 1: **The SPEC sentence processing architecture.** The system consists of the Parser (a simple recurrent network), the Stack (a RAAM network), and the Segmenter (a feedforward network). The gray areas indicate propagation through weights, the solid lines stand for pattern transport, and the dashed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

`chased the cat`. Jain's architecture is similarly limited because of the fixed hardware constraints; XERIC and Sharkey and Sharkey's parser because the RAAM architecture generalizes poorly to new tree structures.

The model described in this paper, SPEC, was especially designed to address the problem of generalization into new sentence structures. SPEC is a descendant of CLAUSES. The central component is the familiar simple recurrent network that reads distributed word representations as its input and generates case-role representations as its output. SPEC's generalization capability is based on simplifying the SRN's task through three architectural innovations: (1) training the SRN to generate a sequence of clause case-role representations as its output (like Stolcke 1990) instead of a single comprehensive representation, (2) introducing a segmenter network that breaks the input sequence into smaller chunks, and (3) introducing a stack network that memorizes constituents over intervening embedded clauses. Below, the SPEC architecture is described in detail, and its performance is demonstrated on an artificially-generated corpus of sentences with complex relative clause structures.

3 The SPEC Architecture

An overview of the architecture is shown in figure 1. The system receives a sequence of word representations as its input, and for each clause in the sentence, forms an output representation indicating the assignment of 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. The collection of case-role representations constitutes the final result of the parse.

SPEC consists of three main components: the Parser, the Segmenter, and the Stack. Below, each component is described in detail and the reasons for the main architectural choices are explained.

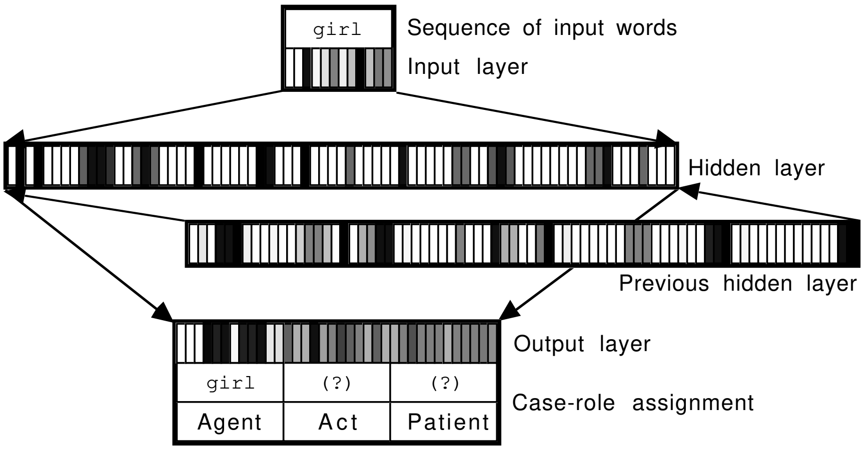

Figure 2: **The Parser network.** The figure depicts a snapshot of the network after it has read the first two words `The` and `girl`. The activity patterns in the input and output assemblies consist of word representations. The input layer holds the representation for the last word, `girl`, and the activity pattern at the output represents the (currently incomplete) case-role assignment of the clause. At this point, it is clear that `girl` is going to be the agent. The act and the patient are not known; the patterns in these slots indicate expectations, that is, averages of all possible alternatives.

3.1 The Parser

The Parser performs the actual transformation of the word sequence into the case-role representations, and like most of the other parsers described above, it is based on the simple recurrent network architecture (figure 2). Words are represented distributively as vectors of gray-scale values between 0 and 1. The component values are initially assigned randomly and modified by the FGREP method (Miikkulainen and Dyer 1989, 1991; Miikkulainen 1993) as part of the learning process. 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 as well (as was done by e.g. McClelland and Kawamoto 1986). SPEC will even work with random word representations, although some of the advantages of distributed representations (such as generalization, robustness, and context representation) would not be as strong.

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[2] in the sentence (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 obvious approach for representing multiple CRVs would be to concatenate them into a single vector at the output of the Parser network. This was the approach taken in CLAUSES

---

[2] The representation was limited to three roles for simplicity. More roles could be included if a richer case-role representation was needed.

(Miikkulainen 1990). Such representation has two serious limitations:

1. The size of the output layer always poses a hard limit on the number of clauses in the sentence. If there is space for three CRVs, sentences with four clauses (such as `The girl saw the boy, who chased the cat, who saw the girl, who liked the dog`) could not be parsed without changing the architecture and retraining the entire network.
2. Somewhat less obviously, such representation turns out to be detrimental to generalization. The network always has to represent the entire sentence in its memory (in the hidden layer). Every new item in the sequence is interpreted in the context of the entire sequence so far. CLAUSES learned to recognize certain sequences of act fragments, and to associate a particular interpretation to each sequence. If there ever was a novel input, such as an additional tail embedding in the end of an otherwise familiar sequence, the network did not know how to combine it with its current hidden-layer representation. As a result, CLAUSES could only process variations of those clause structures it was trained on.

The above problems can be overcome if the network is not required to form a complete sentence representation at its output. Instead, the network generates the CRV for each clause as soon as the information for the clause is complete. Another network (or even a symbolic system such as that of Simmons and Yu 1990) then reads the sequence of complete act representations as its input and builds a representation for the whole sentence using a flexible-size representation technique, such as tensor-product encoding (Dolan 1989; Smolensky 1990).

This is the approach taken in SPEC. 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 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. The early words do in fact fade from the memory as more words are read in, but by itself this effect is not strong enough, and needs to be enforced by an additional network (the Segmenter, discussed in section 3.3). However, even such slight forgetting is strong enough to cause problems with the center embeddings. After parsing `who liked the dog`, the network does not remember that it was `the girl` who `saw the boy`. The system needs a memory component external to the parser so that the top-level parse state can be restored before reading rest of the top-level constituents. This is the task of the Stack network.

3.2 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 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.

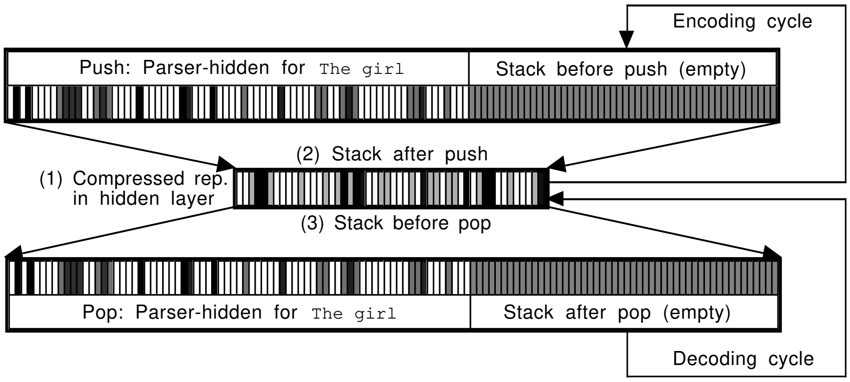

Figure 3: **The Stack network.** This figure simultaneously illustrates three situations that occur at different times during the training and the performance of the Stack: (1) A training situation where the network learns to autoassociate an input pattern with itself, forming a compressed representation 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.

The Stack is implemented as a RAAM network (Pollack 1990) trained to encode and decode linear lists (figure 3). 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.

RAAM networks usually generalize well into encoding and decoding new instances of familiar structures, but poorly into processing new structures (Blank et al. 1992; Chalmers 1990; Chrisman 1992; Sharkey and Sharkey 1992). The deeper the structure, the less accurate its representation, because more and more information will be superimposed on the same fixed-width vector. Fortunately, this is not a major problem for SPEC, because the RAAM network only needs to encode one type of structure (a linear list), and there are very strong memory limitations in human processing of deep embedded structures as well (section 4.3). It should be very easy to train the RAAM network to model human memory for embedded clauses, and it should generalize well to new instances.

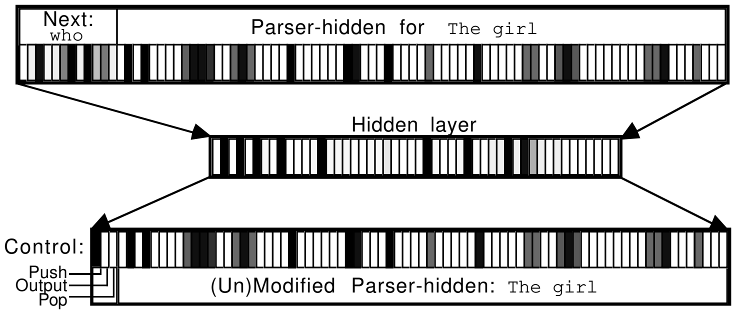

Figure 4: **The Segmenter network.** The Segmenter receives the Parser's hidden-layer pattern as its input together with the next input word, which in this case is `who`. The control outputs are 1, 0, 0, indicating that the Parser's hidden-layer representation should be pushed onto the Stack, the current 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.

3.3 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`. The problem is the same as with CLAUSES: even though the Stack takes care of restoring the earlier state of the parse, the Parser has to learn all the different transitions into the 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. Even though the Parser is free to "forget" the irrelevant information in the early sequence, the hidden-layer patterns remain sufficiently different so that its processing knowledge does not carry over.

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 becomes sufficiently simple so that the entire system can generalize to new structures. The Segmenter plays a central role in the architecture. The next section shows that it is very natural to give the Segmenter a complete control over the entire parsing process.

3.4 Control

At first glance, the control of execution in SPEC seems rather complicated. The activation patterns propagate between networks in a very specific manner, and execution of each network needs to be carefully timed with respect to what the other networks are doing. However, it is actually very easy to train the Segmenter to control the parsing process. The Segmenter always sees the current state of the parse (as encoded in the hidden layer of the Parser network) and the incoming word, and based on this information, it can control the pathways of the system. There are five different control tasks in the SPEC system:

1. Detecting clause transitions and modifying the sequence memory to remove unnecessary previous context as described above.
2. Recognizing the end of the sentence, indicated by a blank (all-0) representation in the input sequence, and subsequently clearing the previous hidden layer (which is all-0 at the beginning of each sentence). This makes it possible for the system to parse multiple sentences without an external "reset".
3. Deciding when to push the Parser's hidden-layer representation onto the stack. This requires opening the pathway from the hidden layer to the "Push" assembly of the Stack, allowing propagation to the Stack's hidden layer, and transporting the resulting pattern back to the Stack's input assembly.
4. Deciding when to pop the previous hidden layer from the stack; this task involves allowing propagation from the Stack's hidden layer to its output layer, transporting the output "Stack" pattern back to its hidden layer, and opening the pathway from the "Pop" assembly to the Parser's previous hidden layer.
5. Deciding when the Parser's output CRV is complete, and consequently, opening the output pathway to the external short-term memory system.

Control is implemented through three additional units at the Segmenter's output (figure 4). These are called Push, Pop, and Output, corresponding to the tasks 3, 4, and 5 above. These units gate the system pathways through multiplicative connections (Pollack 1987; Rumelhart et al. 1986a). The weights on the pathways are multiplied by the output values, so that propagation only takes place when the output is high. The Segmenter is trained to output 1 for the desired propagation, and 0 otherwise.

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.

4 Experiments

A prototype implementation of SPEC was tested with an artificially-generated corpus of relative clause sentences. The purpose was to evaluate the soundness of the basic ideas, test the cognitive

```
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.**

plausibility of the model, and get a feeling for the scale-up possibilities of the approach. The experiments are described below, and some general conclusions drawn from them are presented in the Discussion section.

4.1 Data

The training and testing corpus was generated from a simple phrase structure grammar depicted in table 1. This grammar generates sentences where each clause consists 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` and `the`, the vocabulary consisted of the verbs `chased, liked, saw` and `bit`, and the nouns `boy, girl, dog` and `cat`.

A number of semantic restrictions were imposed on the sentences. A verb could have only certain nouns as its agent and patient (see table 2). These restrictions are not necessary to train SPEC, but they create enough differences in the word usage so that their FGREP representations do not become identical (Miikkulainen and Dyer 1991; Miikkulainen 1993). The main motivation for the restrictions, however, was to determine whether SPEC would be able to use the semantics to aid parsing under difficult conditions. The grammar was used to generate all sentences with up to four clauses, and those that did not match the semantic restrictions were discarded. The final corpus consists of 49 different sentence structures, with a total of 98,100 different sentences (table 3).

Since the SPEC architecture divides the sentence parsing task into low-level pattern transformation, segmentation, and memory, each component needs to see only its own basic constructs during training. The combined architecture then forces generalization into novel combinations of these structures. The Parser and the Segmenter need to be able to process the following three types of sequences:

(1) `The girl saw the boy...` (top level clause)
(2) `...the girl, who saw the boy,...` (`who` as the agent)
(3) `...the girl, who the boy saw,...` (`who` as the patient).

The Segmenter also needs to see four different types of clause transitions, such as

(1) `The girl, who...` (top-level center embedding)
(2) `...the girl, who the boy, who...` (embedded center embedding)

Tmpl.	#-of-s.	Example sentence
1.	20	The girl saw the boy
2.	102	The girl saw the boy, who chased the cat
3.	528	The girl saw the boy, who chased the cat, who saw the girl
4.	2738	The girl saw the boy, who chased the cat, who saw the girl, who liked the dog
5.	2814	The girl saw the boy, who chased the cat, who saw the girl, who the dog bit
* 6.	544	The girl saw the boy, who chased the cat, who the dog bit
7.	2878	The girl saw the boy, who chased the cat, who the dog, who bit the girl, bit
8.	2802	The girl saw the boy, who chased the cat, who the dog, who the girl liked, bit
9.	106	The girl saw the boy, who the dog bit
10.	560	The girl saw the boy, who the dog, who chased the cat, bit
11.	2878	The girl saw the boy, who the dog, who chased the cat, who saw the girl, bit
12.	2962	The girl saw the boy, who the dog, who chased the cat, who the girl chased, bit
13.	544	The girl saw the boy, who the dog, who the girl liked, bit
14.	2898	The girl saw the boy, who the dog, who the girl, who chased the cat, liked, bit
15.	2814	The girl saw the boy, who the dog, who the girl, who the cat saw, liked, bit
16.	106	The girl, who liked the dog, saw the boy
17.	544	The girl, who liked the dog, saw the boy, who chased the cat
18.	2814	The girl, who liked the dog, saw the boy, who chased the cat, who saw the girl
19.	2898	The girl, who liked the dog, saw the boy, who chased the cat, who the dog bit
20.	560	The girl, who liked the dog, saw the boy, who the dog bit
21.	2962	The girl, who liked the dog, saw the boy, who the dog, who chased the cat, bit
22.	2878	The girl, who liked the dog, saw the boy, who the dog, who the boy liked, bit
23.	544	The girl, who liked the dog, who bit the cat, saw the boy
24.	2802	The girl, who liked the dog, who bit the cat, saw the boy, who chased the cat
25.	2878	The girl, who liked the dog, who bit the cat, saw the boy, who the dog bit
26.	2814	The girl, who liked the dog, who bit the cat, who saw the girl, saw the boy
27.	2898	The girl, who liked the dog, who bit the cat, who the boy chased, saw the boy
28.	560	The girl, who liked the dog, who the dog bit, saw the boy
29.	2878	The girl, who liked the dog, who the dog bit, saw the boy, who chased the cat
30.	2962	The girl, who liked the dog, who the dog bit, saw the boy, who the dog bit
31.	2962	The girl, who liked the dog, who the dog, who chased the cat, bit, saw the boy
32.	2878	The girl, who liked the dog, who the dog, who the boy liked, bit, saw the boy
33.	102	The girl, who the dog bit, saw the boy
34.	528	The girl, who the dog bit, saw the boy, who chased the cat
35.	2738	The girl, who the dog bit, saw the boy, who chased the cat, who saw the girl
36.	2814	The girl, who the dog bit, saw the boy, who chased the cat, who the dog bit
37.	544	The girl, who the dog bit, saw the boy, who the dog bit
38.	2878	The girl, who the dog bit, saw the boy, who the dog, who chased the cat, bit
39.	2802	The girl, who the dog bit, saw the boy, who the dog, who the girl liked, bit
*40.	544	The girl, who the dog, who chased the cat, bit, saw the boy
41.	2814	The girl, who the dog, who chased the cat, bit, saw the boy, who liked the girl
42.	2898	The girl, who the dog, who chased the cat, bit, saw the boy, who the girl liked
43.	2802	The girl, who the dog, who chased the cat, who saw the boy, bit, saw the boy
44.	2878	The girl, who the dog, who chased the cat, who the boy chased, bit, saw the boy
45.	528	The girl, who the dog, who the boy liked, bit, saw the boy
46.	2738	The girl, who the dog, who the boy liked, bit, saw the boy, who chased the cat
47.	2814	The girl, who the dog, who the boy liked, bit, saw the boy, who the dog bit
48.	2814	The girl, who the dog, who the boy, who chased the cat, liked, bit, saw the boy
49.	2738	The girl, who the dog, who the boy, who the cat saw, liked, bit, saw the boy
Total	98100	

Table 3: **The sentence structures.** The total number of sentences for each different clause structure is given together with an example sentence. The different clause structures are referred to as "sentence templates" below. SPEC was trained with 100 sentences from templates 6 and 40 each (with complete training of the Stack to up to three levels) and it generalized correctly to all others. Commas are inserted in the examples to help discern the clause boundaries; they were not part of the actual input.

(3) `The girl saw the boy, who...` (top-level tail embedding)
(4) `...the girl, who saw the boy, who...` (embedded tail embedding),

and examples of the two different types of popping operations:

(1) `...the girl, who saw the boy, liked...` (after `who` as agent)
(2) `...the girl, who the boy saw, liked...` (after `who` as patient).

The Stack needs to handle only a very small number of different types of patterns for pushing and popping. Either it receives a center embedding at the top level, followed by a number of center embeddings at deeper levels, such as

(1) `The girl,` (top-level center embedding)

`who the dog,` (first deeper center embedding)
`who the boy,` (second deeper center embedding)
...,

or it receives a number of deeper center embeddings without a preceding top-level embedding:

(2) `The girl saw the boy, who the cat,` (first deeper embedding)
`who the dog,` (second deeper embedding)
...

Because the Segmenter makes all the clause transitions look the same for the Parser, the representations that are pushed on the stack are similar at all levels of embeddings. Therefore, if the Stack is trained to encode, say, a stack of 15 elements, it should generalize to the 16th push without any problems. However, three levels of center embeddings is about the most that would occur in a natural language, and as a result, the architecture cannot really make use of the generalization capabilities of the Stack. The Stack will not generalize to encoding and decoding a 3-element stack after it has been trained only up to 2-element stacks, and there is little point in doing that anyway. It is quite easy to train the Stack to up to 3 levels of embeddings and thereby guarantee that the Stack is not going to be limiting the generalization capabilities of the system.

The modular structure of SPEC makes it possible to train it efficiently with a minimal set of training examples as described above. On the other hand, it is unclear how such training would map to human learning of language. While the case-role targets could correspond to e.g. visual images associated with the sentence input, it is not easy to see how the targets for the segmenter, for example, could be generated by the language acquisition system. Therefore, as it currently stands, SPEC should be seen primarily as a model of human performance rather than learning. Learning is essential for putting the model together, and the current implementation shows one way of doing it efficiently. Other, cognitively more valid learning mechanisms may be possible, and will be studied in future research.

4.2 Training Methodology

There is a variety of strategies for training a modular system such as SPEC. They usually lead to comparable results, but vary in amount of computational and programming effort involved, final accuracy, and robustness of the trained system.

One possibility is to train the entire SPEC as a whole, propagating the patterns between modules as during normal performance. For example, the output of the Stack would be propagated into the previous-hidden-layer assembly of the Parser as it is, even if it is highly inaccurate during early training. The advantage is that the modules learn to compensate for each other's errors, and final accuracy may be better. On the other hand, convergence is often slower, because the modules have to continuously adjust to each other's changing output representations.

If SPEC is to be trained as a whole, a set of templates from table 3 must be selected so that all the basic constructs are included in the set of sentences. One such set consists of templates 3, 15, and 49. Indeed, trained with 100 randomly chosen examples from each template, the network correctly generalized to all other sentences in the entire corpus.

On the other hand, each component can be trained separately, with compatible training data from the same set of examples but without propagating the actual output to the input of the next network. For example, after the previous-hidden-layer representation is obtained from the stack, it is cleaned up (i.e. replaced by the correct representation) before actually loading it into the previous hidden layer. This way the modules learn more independently, and converge faster. If the Parser is trained first, the Segmenter and the Stack can be trained very efficiently with the Parser's final hidden-layer patterns. The total training time in CPU cycles is minimized this way. It is also possible to train the different networks simultaneously on separate machines, thereby minimizing the wallclock training time. In the end, after the networks have learned to produce output close to their targets, they can be connected and they will work well together, even filter out each other's noise (Miikkulainen 1993).

Training SPEC is not computationally very intensive with this particular corpus, and therefore, the most convenient training strategy was selected for the experiments reported below. All modules were trained separately and simultaneously on a single machine, sharing the gradually evolving word and hidden-layer representations. With this strategy, it is enough to train SPEC only with templates 6 and 40, because they contain all the basic constructs for the Parser and the Segmenter. Complete training data for the Stack can be obtained from Parser's hidden layer during the course of processing sentences 6 and 40.

4.3 Results

The word representations consisted of 12 units. Parser's hidden layer was 75 units wide, that of the Segmenter 50 units, and that of the Stack 50 units. All networks were trained with plain on-line backpropagation with 0.1 learning rate and without momentum. The training set consisted of 100 randomly-selected sentences from templates 6 and 40 each. Both the Parser and the Segmenter developed word representations at their input layers (with a learning rate of 0.001). The Stack was trained to encode and decode up to three levels of center embeddings.

The convergence was very strong. After 400 epochs, the average error per output unit was 0.019 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, developed by FGREP, 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 0.019 for the Parser, 0.009 for the Segmenter (0.002 for control), and 0.005 for the Stack. There was very little variation between templates 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 templates, but to new templates as well, which the earlier sentence processing architectures such as CLAUSES 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?

There are two ways to elicit enough errors from SPEC to analyze its limitations: (1) it can be tested during early training, or (2) its memory can be disturbed by noise. In a sense, testing during training illustrates developmental effects, whereas adding noise can be claimed to simulate overload, stress, cognitive impairment, and lack of concentration situations. Both methods produce similar results; ones obtained with noise are reported below.

The Stack's performance was degraded by adding 30% noise in its propagation. During encoding, the final value h_i of the hidden unit i was obtained from r_i, the value after correct propagation, by the transformation

$$h_i = 0.70 r_i + 0.30 X, \tag{1}$$

where X is a random variable uniformly distributed within [0, 1]. Similarly during decoding, the output values o_i were degraded by

$$o_i = 0.70 c_i + 0.30 X, \tag{2}$$

where c_i is the correct value of unit i. The SPEC system turned out to be remarkably robust against such degradation. 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` (template 48), SPEC would have 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 (figure 5). For example, SPEC could usually connect `boy` with `liked`, but it was harder for it to remember that it was the `dog` who `bit` and the `girl` who `saw` 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 (Blaubergs and Braine 1974; Blumenthal and Boakes 1967; Foss and Cairns 1970; Larkin and Burns 1977; Miller and Isard 1964; Schlesinger 1968). 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. With the example data, the percentage of correct agents after the first pop is always around 80%, whether that pop occurs after a single embedding (as in template 16), two embeddings (as in 40), or three (as in 48/49, figure 5).

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 13,650). 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. The weights of the Parser network have learned to encode this constraint. 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

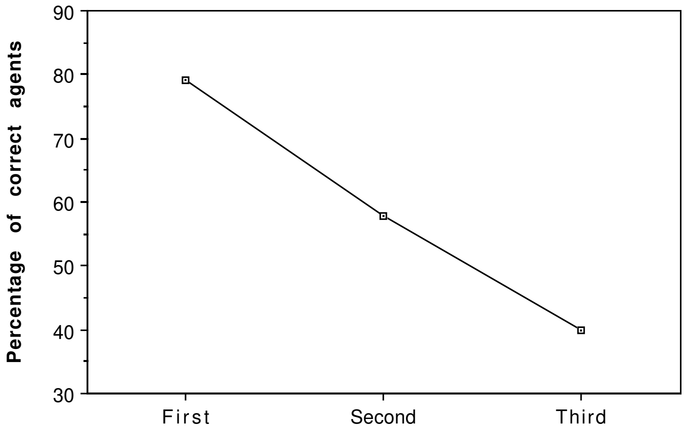

Figure 5: **Memory accuracy after return from center embeddings (with 30% noise degradation).** The percentage of correctly-remembered agents is plotted after the first, second, and the third pop in sentence templates 48 and 49 (represented by the words `boy`, `dog` and `girl` in the example sentences of table 3). Each successive pop is harder and harder to do correctly (with statistical significance t=9.7; df=11,102; p=10^{-22}, and t=13.1; df=11,102; p=10^{-39}). Similarly, SPEC remembers about 84% of the patients correctly after the first pop, and 67% after the second pop.

choice would give only 54%[3]. 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 (figure 6). 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`.

A similar effect has been observed in human processing of relative clause structures. About 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 and Zurif (1976) observed similar behavior in aphasics. This effect is often attributed to limited capability for processing syntax. The SPEC experiment indicates 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.

---

[3] The difference is statistically significant with t=8.1; df=48; p=10^{-10}.

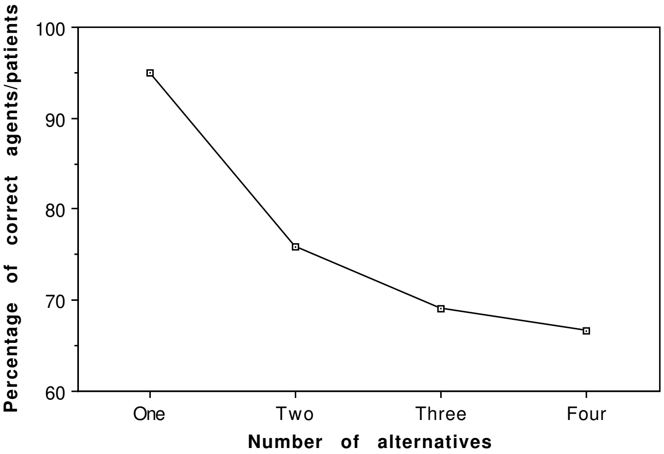

Figure 6: **Effect of the semantic restrictions on the memory accuracy (with 30% noise degradation).** The percentage of correctly-remembered agents and patients over the entire corpus is plotted against how strongly they were semantically associated with the verb. When there was only one alternative (such as `dog` as an agent for `bit` or `cat` as the patient of `chased`), SPEC remembered 95% of them correctly. There was a marked drop in accuracy with two, three and four alternatives (with significance $t{=}878.3$; $df{=}106,982$; $p \approx 0$, $t{=}75.3$; $df{=}151,334$; $p \approx 0$, and $t{=}28.9$; $df{=}106,982$; $p \approx 0$).

5 Discussion

SPEC is quite insensitive to configuration and simulation parameters. Many variations were tried in the experiments, such as hidden layers with 10–75 units, training sets with 200–4,000 sentences, different templates for training, modifying word representations in the Parser only, not modifying them at all, fixed learning rates 0.1–0.001 for weights and representations, gradually reducing the learning rates, training the modules together, and training them separately. All these variations led to comparable results. Such flexibility suggests that the approach is very strong, and there should be plenty of room for adapting it to more challenging experiments.

Several other observations also indicate that the approach 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. Finally, with the FGREP mechanism and the ID+content technique (section 2), it is possible to automatically form meaningful distributed representations for a large number of words, even to acquire them incrementally, and the network will know how to process them in new situations (Miikkulainen and Dyer 1991; Miikkulainen 1993). Taken together, the above three points suggest that strong systematicity in the sense of Hadley (1992; see also Fodor and Pylyshyn 1988) is possible in distributed connectionist networks. SPEC with the ID+content technique would be able to parse new sentence structures, new combinations of familiar words, and also new words in a systematic manner similar to human language users.

The most immediate direction for future work is to apply the SPEC architecture to a wider variety of grammatical constructs and to larger vocabularies. Two main issues need to be addressed in this work:

1. It will be necessary to develop methods for representing the final parse result. Currently, SPEC passes the output CRVs to an unspecified short-term memory system. This system needs to be made an explicit part of SPEC, preferably in such a way that the sentence representation can be used by other subsymbolic networks in processing multi-sentential text and in various reasoning tasks.
2. It might be possible to utilize the interpolation capability and context sensitivity of distributed neural networks at the level of processing structure. The current SPEC architecture generalizes to new instances of basic constructs, but generalization to new sentence structures is built in into the architecture. Perhaps a way can be found to generalize also at the level of control and segmentation. This way, the system could perform more robustly when the input is irregular (or ungrammatical), and contains novel basic constructs.

The Segmenter is perhaps the most significant new feature of the SPEC architecture. Most connectionist systems to date are based on simple propagation through homogenous networks or between networks of a modular system. As we have seen above, such systems are very good at dealing with regularities and integrating large amounts of small pieces of evidence, but they do not easily lend themselves to processing complex knowledge structures and unusual and novel situations. Such systems are not "conscious" of what they are doing, that is, they do not have representations concerning the nature of their internal representations and processes. As a result, they cannot employ high-level strategies in controlling the execution; their behavior is limited to a series of reflex responses.

With a comprehensive high-level monitor and control system, it would be possible to build much more powerful subsymbolic models. Current systems try to process every input in exactly the same way, regardless of whether the input makes sense or not. A high-level controller could monitor the feasibility of the task and the quality of the output, and initiate exception processing when the usual mechanisms fail. For example, unusual events or ungrammatical input could be detected and then processed by special mechanisms. The monitor could also clean up internal inaccuracies and keep the system execution on a stable path. Sequential high-level procedures and reasoning mechanisms could be implemented, such as comparing alternative interpretations and applying high-level rules to conclude new information. Equipped with such mechanisms, subsymbolic models would be able to perform much more robustly in the real world. Eventually, the goal would be to develop a distributed control system that would act as a high-level "conscious" monitor, similar to the central executive system in psychological and neuropsychological theories of controlled processes (Baddeley 1986; Cowan 1988; Logan and Cowan 1984; Norman and Shallice 1980; Posner and Snyder 1975; Schneider and Shiffrin 1977; Shallice 1982, 1988; Shiffrin and Schneider 1977, 1984).

The Segmenter is a first step toward implementing such a control system in the connectionist framework (see also Jacobs et al. 1991; Jain 1991; Schneider and Detweiler 1987; Sumida 1991). This module 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. Further developing such control mechanisms in parsing and in other cognitive tasks constitutes a most exciting direction for future research.

6 Conclusion

Much of the motivation for SPEC comes from the artificial intelligence point of view, that is, by the desire to build a system that (1) is able to process nontrivial input like symbolic systems, and (2) makes use of the unique properties of distributed neural networks such as learning from examples, spontaneous generalization, robustness, context sensitivity, and integrating statistical evidence. While SPEC does not address several fundamental issues in connectionist natural language processing (such as processing exceptions and representing flexible structure), it goes a long way in showing 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 first implementation of a central executive in generating high-level behavior opens exciting possibilities for future work.

Note

The code and data for the SPEC experiments described in this paper are available through the World Wide Web at URL `http://www.cs.utexas.edu/users/nn`, and also by anonymous ftp from `cs.utexas.edu:pub/neural-nets/spec`.

Acknowledgements

Special thanks go to Dennis Bijwaard for the initial implementation and experiments that led to the SPEC architecture. Most of the simulations were run on the CRAY Y-MP 8/864 and the IBM RS6000s at the University of Texas Center for High-Performance Computing.

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.

Allen, R. B., and Riecken, M. E. (1989). Reference in connectionist language users. In Pfeifer, R., Schreter, Z., Fogelman Soulié, F., and Steels, L., editors, *Connectionism in Perspective*, 301–308. New York: Elsevier.

Baddeley, A. D. (1986). *Working Memory.* Oxford, UK; New York: Oxford University Press.

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.

Blank, D. S., Meeden, L. A., and Marshall, J. B. (1992). Exploring the symbolic/subsymbolic continuum: A case study of RAAM. In Dinsmore, J., editor, *The Symbolic and Connectionist Paradigms: Closing the Gap*, 113–148. Hillsdale, NJ: Erlbaum.

Blaubergs, M. S., and Braine, M. D. S. (1974). Short-term memory limitations on decoding self-embedded sentences. *Journal of Experimental Psychology*, 102:745–748.

Blumenthal, A. L., and Boakes, R. (1967). Prompted recall of sentences. *Journal of Verbal Learning and Verbal Behavior*, 6:674676.

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.

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

Chrisman, L. (1992). Learning recursive distributed representations for holistic computation. *Connection Science*, 3:345–366.

Cook, W. A. (1989). *Case Grammar Theory.* Washington, DC: Georgetown University Press.

Cosic, C., and Munro, P. (1988). Learning to represent and understand locative prepositional phrases. In *Proceedings of the 10th Annual Conference of the Cognitive Science Society*, 257–262. Hillsdale, NJ: Erlbaum.

Cowan, N. (1988). Evolving conceptions of memory storage, selective attention, and their mutual constraints within the human information-processing system. *Psychological Bulletin*, 104:163–191.

Dolan, C. P. (1989). *Tensor Manipulation Networks: Connectionist and Symbolic Approaches to Comprehension, Learning and Planning.* PhD thesis, Computer Science Department, University of California, Los Angeles. Technical Report UCLA-AI-89-06.

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

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

Elman, J. L. (1991b). Incremental learning, or The importance of starting small. In *Proceedings of the 13th Annual Conference of the Cognitive Science Society*, 443–448. Hillsdale, NJ: Erlbaum.

Fillmore, C. J. (1968). The case for case. In Bach, E., and Harms, R. T., editors, *Universals in Linguistic Theory*, 0–88. New York: Holt, Rinehart and Winston.

Fodor, J. A., and Pylyshyn, Z. W. (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28:3–71.

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.

Hadley, R. F. (1992). Compositionality and systematicity in connectionist langauge learning. In *Proceedings of the 14th Annual Conference of the Cognitive Science Society*, 659–664. Hillsdale, NJ: Erlbaum.

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.

Larkin, W., and Burns, D. (1977). Sentence comprehension and memory for embedded structures. *Memory and Cognition*, 5:17–22.

Lee, G., Flowers, M., and Dyer, M. G. (1990). Learning distributed representations of conceptual knowledge and their application to script-based story processing. *Connection Science*, 2:313–346.

Logan, G. D., and Cowan, W. B. (1984). On the ability to inhibit thought and action: A theory of an act of control. *Psychological Review*, 91:295–327.

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. (1990). A PDP architecture for processing sentences with relative clauses. In Karlgren, H., editor, *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. (1989). Encoding input/output representations in connectionist cognitive systems. In Touretzky, D. S., Hinton, G. E., and Sejnowski, T. J., editors, *Proceedings of the 1988 Connectionist Models Summer School*, 347–356. San Mateo, CA: Morgan Kaufmann.

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.

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

Norman, D. A., and Shallice, T. (1980). Attention to action: Willed and automatic control of behavior. Technical Report 99, Center for Human Information Processing, University of California, San Diego.

Pollack, J. B. (1987). Cascaded back-propagation on dynamic connectionist networks. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, 391–404. Hillsdale, NJ: Erlbaum.

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

Posner, M. I., and Snyder, C. R. (1975). Attention and cognitive control. In Solso, R. L., editor, *Information Processing and Cognition*, 55–85. Hillsdale, NJ: Erlbaum.

Rumelhart, D. E., Hinton, G. E., and McClelland, J. L. (1986a). A general framework for parallel distributed processing. In Rumelhart, D. E., and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, 45–76. Cambridge, MA: MIT Press.

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

Schlesinger, I. M. (1968). *Sentence Structure and the Reading Process.* The Hague: Mouton.

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

Schneider, W., and Shiffrin, R. M. (1977). Controlled and automatic human information processing I: Detection, search, and attention. *Psychological Review*, 84:1–66.

Servan-Schreiber, D., Cleeremans, A., and McClelland, J. L. (1989). Learning sequential structure in simple recurrent networks. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 1*, 643–652. San Mateo, CA: Morgan Kaufmann.

Servan-Schreiber, D., Cleeremans, A., and McClelland, J. L. (1991). Graded state machines: The representation of temporal contingencies in simple recurrent networks. *Machine Learning*, 7:161–194.

Shallice, T. (1982). Specific impairments of planning. *Philosophical Transactions of the Royal Society of London B*, 298:199–209.

Shallice, T. (1988). *From Neuropsychology to Mental Structure.* Cambridge, UK: Cambridge University Press.

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.

Shiffrin, R. M., and Schneider, W. (1977). Controlled and automatic human information processing II: Perceptual learning, automatic attending, and a general theory. *Psychological Review*, 84:127–190.

Shiffrin, R. M., and Schneider, W. (1984). Automatic and controlled processing revisited. *Psychological Review*, 91:269–276.

Simmons, R. F., and Yu, Y.-H. (1990). Training a neural network to be a context-sensitive grammar. In *Proceedings of the Fifth Rocky Mountain Conference on Artificial Intelligence, Las Cruces, NM*, 251–256.

Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence*, 46:159–216.

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. (1989). Applying contextual constraints in sentence comprehension. In Touretzky, D. S., Hinton, G. E., and Sejnowski, T. J., editors, *Proceedings of the 1988 Connectionist Models Summer School*, 338–346. San Mateo, CA: Morgan Kaufmann.

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.

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., 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.

Weckerly, J., and Elman, J. L. (1992). A PDP approach to processing center-embedded sen