A PDP ARCHITECTURE FOR PROCESSING SENTENCES WITH RELATIVE CLAUSES [*][†]

Risto Miikkulainen
Artificial Intelligence Laboratory, Computer Science Department
University of California, Los Angeles, CA 90024
risto@cs.ucla.edu

Abstract

A modular parallel distributed processing architecture for parsing, representing and paraphrasing sentences with multiple hierarchical relative clauses is presented. A lowel-level network reads the segments of the sentence word by word into partially specified case-role representations of the acts. A higher-level network combines these representations into a list of complete act representations. This internal representation stores the information conveyed by the sentence independent of its linguistic form. The information can be output in natural language in different form or style, e.g. as a sequence of simple sentences or as a complex sentence consisting of relative clauses. Generating output is independent from parsing, and what actually gets generated depends on the training of the generator modules.

1 Introduction

Parsing a sentence means reading the input text into an internal representation, which makes the relations of the constituents explicit. In symbolic parsing, the result is usually a semantic network structure, e.g. a conceptual dependency representation [17; 3], or a syntactic parse tree augmented with semantic restrictions [13; 9]. The advantage of this approach is that sentences with arbitrary complexity can be parsed and represented. However, processing knowledge must be hand-coded with specific examples in mind. Rules for expectations, defaults and generalizations must be explicitly programmed.

The localist connectionist models [2; 20; 5; 1; 19; 7] provide more general mechanisms for inferencing and give a more plausible account of the parsing process in terms of human performance. However, these networks need to be carefully crafted for each example.

The main advantage of the distributed connectionist approach [8; 18; 12] is that processing is learned from examples. Expectations about unspecified constituents arise automatically from the processing mechanism, and generalizations into new inputs result automatically from the representations. Any statistical regularity in the training examples is automatically utilized in making inferences. The result in distributed parsing at the sentence level is e.g. an assembly-based case-role representation of the sentence [8; 10]. The output layer of the network is divided into partitions, each representing a case role, and distributed activity patterns in the assemblies indicate the words filling these roles.

Representing complex structures is problematic in the distributed approach [6; 15]. The proposed sentence processing architectures can only deal with simple, straightforward sentences. Case-role analysis is feasible only when the sentences consist of single acts, so that unique case role can be assigned for each constituent. The approach can be extended and roles reserved for attributes of the constituents also. However, sentences with relative clauses remain intractable.

A hierachical PDP architecture for parsing, representing and paraphrasing sentences with multiple hierarchical relative clauses is described in this paper. Each relative clause is itself an act, and has its own case-role representation. The whole sentence is represented as a *collection of these acts*. The relations of the acts are implicit in their content, rather than explicit in the structure of the representation. The original complex hierarchical sentence, as well as a simplified paraphrase of it, can be produced from the list of the act representations.

2 System architecture

2.1 Overview

The system consists of four hierarchically organized subnetworks (figure 1). The act parser reads the input words one at a time, and forms a stationary case-role representation for each act fragment (defined as part of sentence separated by commas). The sentence parser reads these case-role representations one at a time, and forms a stationary representation of the whole sentence as its output. This is the internal representation of the sentence.

The sentence generator takes the internal representation as its input, and produces a sequence of case-role representations of the act fragments as its output. These are fed one at a time to the act generator, which generates the sequence of words for each act fragment. During performance, the four networks are connected in a chain, the output of one network feeding the input of another (figure 1). During training, each network is trained separately with compatible

[*] This research was supported in part by a grant from the ITA Foundation, and in part by grants from the Academy of Finland, the Emil Aaltonen Foundation, the Foundation for the Advancement of Technology, and the Alfred Kordelin Foundation (Finland). The simulations were carried out on the Cray X-MP/48 at the San Diego Supercomputer Center.

[†] To appear in the *Proceedings of the 13th International Conference on Computational Linguistics (COLING-90)*

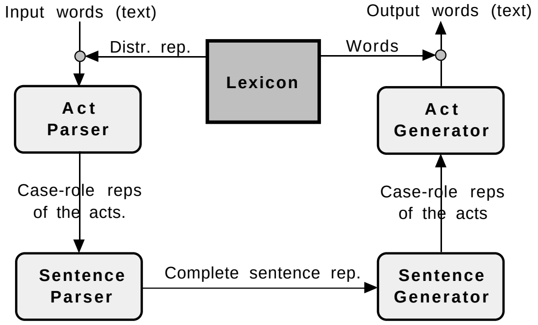

Figure 1: **Overview of the model.** The model consists of parsing and generating subsystems, and a central lexicon of distributed word representations. Each subsystem consists of two hierarchically organized modules, with the case-role assignment of the act as an intermediate representation.

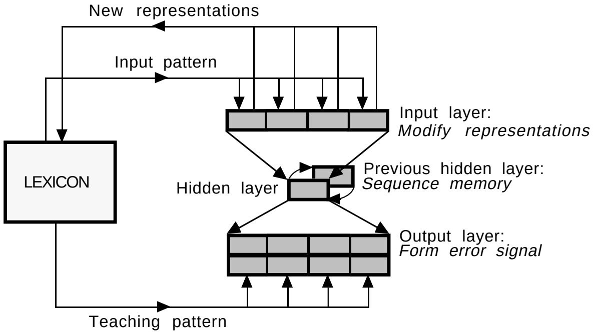

Figure 3: **Recurrent FGREP-module.** At the end of each backpropagation cycle, the current input representations are modified at the input layer according to the error signal. The new representations are loaded back to the lexicon, replacing the old ones.

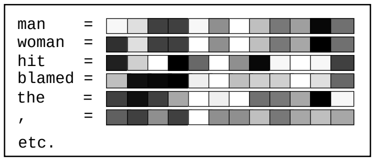

Figure 2: **Lexicon.** The lexicon is an associative memory, associating the text form of each word with its distributed representation. The representation is a vector of real numbers between 0 and 1, shown as grey-scale values from white to black.

I/O data (figure 6).

The input/output of each network is composed of distributed representations of words. These representations are stored in a central lexicon (figure 2), and all networks use the same representations. Each network is a Recurrent FGREP module, i.e. a three-layer backpropagation network with sequential input or output, which develops the word representations automatically while it is learning the processing task.

2.2 Recurrent FGREP - A building block

The FGREP mechanism (Forming Global Representations with Extended backPropagation) [10; 12] is based on a basic three-layer backward error propagation network (figure 3). The network learns the processing task by adapting the connection weights according to the standard backpropagation equations [16, pages 327-329]. At the same time, representations for the input data are developed at the input layer according to the error signal extended to the input layer. Input and output layers are divided into assemblies and several items are represented and modified simultaneously.

The representations are stored in an external lexicon network. A routing network forms each input pattern and the corresponding teaching pattern by concatenating the lexicon entries of the input and teaching items. Thus the same representation for each item is used in different parts of the backpropagation network, both in the input and in the output.

The process begins with a random lexicon containing no pre-encoded information. During the course of learning, the *representations adapt to the regularities of the task*. It turns out that single units in the resulting representation do not necessarily have a clear interpretation. The representation does not implement a classification of the item along identifiable features. In the most general case, the representations are simply profiles of continuous activity values over a set of processing units. This representation pattern as a whole is meaningful and can be claimed to code the meaning of that word. The representations for words which are used in similar ways become similar.

Recurrent FGREP [11; 12] is an extension of the basic FGREP architecture to sequential input and output, based on [4]. A copy of the hidden layer at time step t is saved and used along with the actual input at step $t+1$ as input to the hidden layer (figure 3). The previous hidden layer serves as a sequence memory, essentially remembering where in the sequence the system currently is and what has occurred before. During learning, the weights from the previous hidden layer to the hidden layer proper are modified as usual according to the backpropagation mechanism.

The Recurrent FGREP module can be used for reading a sequence of input items into a stationary output representation, or for generating an output sequence from a stationary input. In a *sequential input network*, the actual input changes at each time step, while the teaching pattern stays the same. The network is forming a stationary representation of the sequence. In a *sequential output network*, the actual input is stationary, but the teaching pattern changes at each step. The network is producing a sequential interpretation of its input. The error is backpropagated and weights are changed at each step. Both types of Recurrent FGREP networks develop representations in their input layers.

2.3 Connecting the building blocks in the performance phase

Let us present the system with the following sentence: `The woman, who helped the girl, who the boy hit, blamed the man`.

The task of the act parser network (figure 4) is to form a stationary case-role representation for each part of the sentence, for complete acts (`who helped the girl` and `who the boy hit`) as well as for act fragments (`the woman` and `blamed the man`). There is an assembly of units at the output layer of this net-

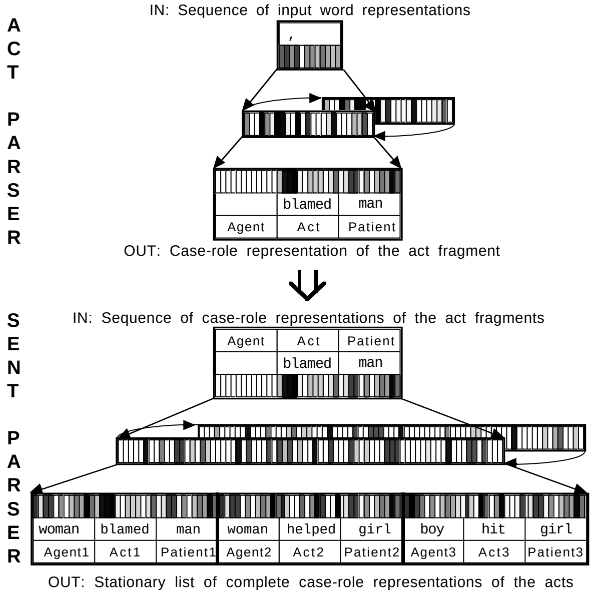
A C T P A R S E R
IN: Sequence of input word representations
,
blamed
man
Agent
Act
Patient
OUT: Case-role representation of the act fragment
S E N T P A R S E R
IN: Sequence of case-role representations of the act fragments
Agent
Act
Patient
blamed
man
woman
blamed
man
woman
helped
girl
boy
hit
girl
Agent1
Act1
Patient1
Agent2
Act2
Patient2
Agent3
Act3
Patient3
OUT: Stationary list of complete case-role representations of the acts

Figure 4: **Networks parsing the sentence.** Snapshot of the simulation after the whole sentence `The woman, who helped the girl, who the boy hit, blamed the man` has been read in. The output of the act parser shows the case-role representation of the last act fragment, `blamed the man`. The output of the sentence parser displays the result of the parse, the internal representation of the whole sentence.

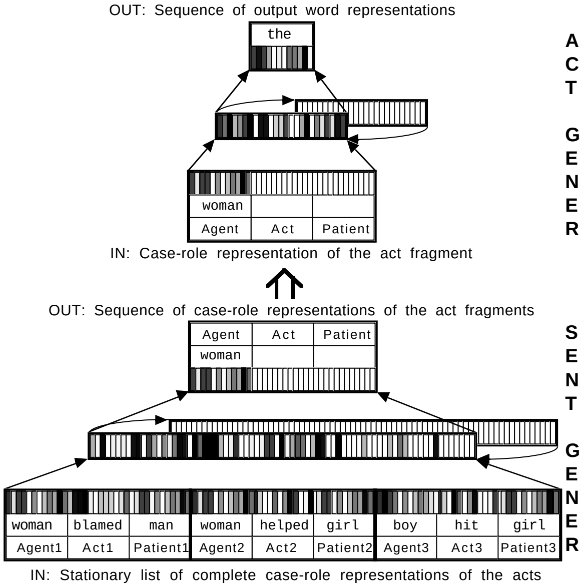
OUT: Sequence of output word representations
the
A C T G E N E R
woman
Agent
Act
Patient
IN: Case-role representation of the act fragment
OUT: Sequence of case-role representations of the act fragments
Agent
Act
Patient
woman
S E N T G E N E R
woman
blamed
man
woman
helped
girl
boy
hit
girl
Agent1
Act1
Patient1
Agent2
Act2
Patient2
Agent3
Act3
Patient3
IN: Stationary list of complete case-role representations of the acts

Figure 5: **Networks generating the sentence.** The system is in the beginning of generating `The woman, who helped the girl, who the boy hit, blamed the man`. The sentence generator has produced the case-role representation of the first act fragment, `The woman,` and the act generator has output the first word of that fragment. The previous hidden layers are blank during the first step.

work for each case role. Each assembly is to be filled with the distributed activity pattern of the word that fills that role. For example, the correct representation for `who the boy hit` is agent=`boy`, act=`hit` and patient`who`.

As each word is read, its distributed representation is obtained from the lexicon, and loaded into the input layer of the act parser network. The activity propagates through the network, and a distributed pattern forms at the output layer, indicating expectations about possible act representations at that point. The activity pattern at the hidden layer is copied to the previous-hidden-layer assembly, and the next word is loaded into the input layer. *Each successive word narrows down the possible interpretations*, and the case-role representation of a specific act gradually forms at the output.

After reading `the` and `woman`, the network knows to generate the pattern for `woman` in the agent assembly, because in our training examples, the first noun is always the agent. The pattern in the act assembly is an average of `helped` and `blamed`, the two possible acts for `woman` in our data. The pattern in the patient assembly is an average of all patients for `helped` and `blamed`. Reading a verb next would establish the appropriate representation in the act assembly, and narrow down the possibilities in the patient assembly.

However, the network reads a comma next, which means that the top-level act is interrupted by the relative clause (commas separate clause fragments in our data; a way to segment clauses without commas is outlined in section 4). The network is trained to clear the expectations in the unspecified assemblies, i.e. to form an incomplete case-role interpretation of the top-level act so far. This representation is passed on as the first input to the sentence parser module.

The act parser then goes on to parse the relative clause `who helped the girl` independently from what it read before, i.e. the pattern in its previous-hidden-layer assembly is cleared before reading `who`. The complete case-role representation of the relative clause is passed on to the sentence parser as its second input. Similarly, `who the boy hit` is parsed and its representation passed on to the sentence parser. The act parser then receives the rest of the top-level act, `blamed the man`, which is again parsed independently, and its incomplete case-role representation (figure 4) passed on to the sentence parser.

The sentence parser reads the sequence of these four case-role representations, combines the incomplete case-role representations into a complete representation of the top-level act, and determines the referents of the `who` pronouns. The result is a list of three completely specified case-role representations, `|woman blamed man|`, `|woman helped girl|` and `|boy hit girl|` (bottom of figure 4).

The list of case-role representations is the final result of the parse, the internal representation of the sentence. It is a *canonical representation with all the structural information coded into simple acts.* All information is accessible in parallel, and can be directly used for further processing.

The output side of the system (figure 5) demonstrates how the information in the internal representation can be output in different ways in natural language. The output process is basically the reverse of the reading process. The sentence generator network takes the internal representation as its input and produces the case-role representation of the first act fragment, |woman (blank) (blank)| as its output (figure 5). This is fed to the act generator, which generates the distributed representation of the, the first word of the act fragment. The representation in the lexicon closest to the output pattern is obtained, and the text form of that entry is put into the stream of output text.

The hidden layer pattern of the word generator is copied into its previous-hidden-layer assembly, and the next word is output. The commas segment the output as well. As soon as a comma is output, the sentence generator network is allowed to generate the case-role representation of the next act fragment.

The sentence generator can produce different output versions from the same internal representation, depending on its training. (1) The acts can be output sequentially one at a time as separate simple sentences, or (2) a single output sentence with a complex relative clause structure can be generated. The point is that it does not matter how the internal representation is arrived at, i.e. whether it was read in as a single sentence, as several sentences, or maybe produced as a result of a reasoning process. *Generating output sentences is independent from parsing, and the form and style of the output depends on the processing knowledge of the sentence generator.*

In case (1) the sentence generator produces the case-role representations |woman blamed man|, |woman helped girl| and |boy hit girl|, and the act generator generates The woman blamed the man, The woman helped the girl, The boy hit the girl. In case (2) the sentence generator produces the sequence |woman (blank) (blank)|, |who helped girl|, |boy hit who|, |(blank) blamed man|, and the output text reads The woman, who helped the girl, who the boy hit, blamed the man.

2.4 Training phase

A good advantage of the modular architecture can be made in training the networks. The tasks of the four networks are separable, and they can be trained separately as long as compatible I/O material is used. The networks must be trained simultaneously, so that they are always using and developing the same representations (figure 6).

The lexicon ties the separated tasks together. Each network modifies the representations to improve its performance in its own task. The pressure from other networks modifies the representations also, and they evolve slightly differently than would be the most efficient for each network independently. The networks compensate by adapting their weights, so that in the end the representations and the weights of all networks are in harmony. The requirements of the different tasks are combined, and the final representations reflect the total use of the words.

If the training is successful, the output patterns produced by one network are exactly what the next network learned to process as its input. But even if the learning is less than complete, the networks perform well together. Erroneous output patterns are noisy input to the next network, and neural networks in general tolerate, even filter out noise very efficiently.

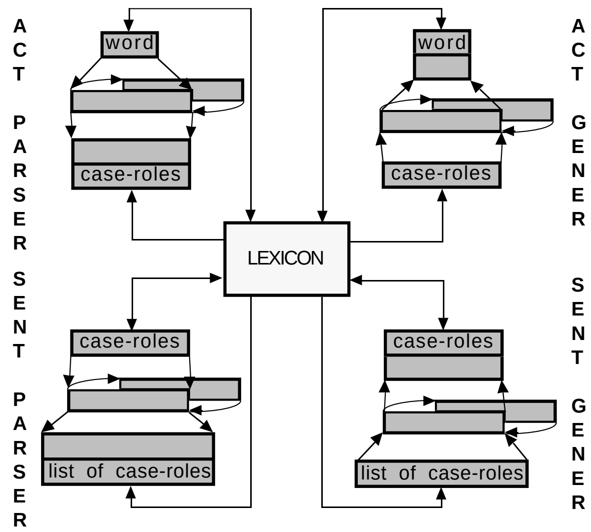

Figure 6: **Training configuration.** Each network is trained separately and simultaneously, developing the same lexicon.

3 Experiments

3.1 Training data

The system was trained with sentences generated using the 17 templates shown in table 1. The acts consisted of three case-roles: agent, the act (i.e. the verb), and patient. A relative clause could be attached to the agent or to the patient, and these could fill the role of the agent or the patient in the relative clause.

Certain semantic restrictions were imposed on the templates to obtain more meaningful sentences. The restrictions also create enough differences in the usage of the words, so that their representations do not become identical (see [12]). A verb could have only specified nouns as its agent and patient, listed in table 2. Sentences with two instances of the same noun were also excluded. With these restrictions, the templates generate a total of 388 sentences. All sentences were used to train the system. Generalization was not studied in these experiments (for a discussion of the generalization capabilities of FGREP systems see [12]).

Two different versions of the sentence generator were trained: one to produce the output as a sequence of simple sentences, and another to produce a single sentence with hierarchical relative clauses, i.e. to reproduce the input sentence. The act generator was trained only with the act fragments from the complex sentences. Because these contain the simple acts, the act generator network effectively learned to process the ouput of the first version of the sentence generator as well.

The four networks were trained separately and simultaneously with compatible I/O data. This means that the output patterns, which are more or less incorrect during training, were not directly fed into the next network. They were replaced by the correct patterns, obtained by concatenating the current word representations in the lexicon. The word representations consisted of 12 units, the hidden layers of the act networks of 25 units, and the hidden layers of the

Tp.	N.	Example sentence
1.	12	The woman blamed the man
2.	24	The woman blamed the man, who hit the girl
3.	20	The woman blamed the man, who hit the girl, who blamed the boy
4.	24	The woman blamed the man, who hit the girl, who the boy hit
5.	20	The woman blamed the man, who the girl blamed
6.	28	The woman blamed the man, who the girl, who blamed the boy, blamed
7.	24	The woman blamed the man, who the girl, who the boy hit, blamed
8.	24	The woman, who helped the boy, blamed the man, who helped the girl
9.	28	The woman, who helped the boy, blamed the man, who the girl blamed
10.	20	The woman, who the boy hit, blamed the man, who helped the girl
11.	24	The woman, who the boy hit, blamed the man, who the girl blamed
12.	24	The woman, who helped the girl, blamed the man
13.	24	The woman, who helped the girl, who blamed the boy, blamed the man
14.	28	The woman, who helped the girl, who the boy hit, blamed the man
15.	20	The woman, who the boy hit, blamed the man
16.	24	The woman, who the boy, who hit the girl, hit, blamed the man
17.	20	The woman, who the boy, who the girl blamed, hit, blamed the man

Table 1: **Sentence templates.**

Verb	Case-rl	Nouns
helped	Agent:	man,woman
	Patient:	boy,girl
hit	Agent:	man,boy
	Patient:	woman,girl
blamed	Agent:	woman,girl
	Patient:	man,boy

Table 2: **Restrictions.**

There are 3 different verbs, with 2 possible agents and patients each (table 2). These words are used to generate sentences with the 17 different sentence templates (table 1). The same noun cannot occur in two places in the same sentence. An example sentence for each template is given, together with the number of different sentences the template generates.

Network	Words	0.15	E_{avg}
Act parser	100	100	.027
Sentence parser	93	86	.083
Sentence gener(simple)	100	96	.047
→ Act generator	100	98	.039
Sentence gener(clauses)	98	87	.071
→ Act generator	97	92	.060

Table 3: **Performance.** The first column indicates the percentage of correct words out of all output words. The second column indicates the percentage of output units which were within 0.15 of the correct value, and the last column shows the average error per output unit.

sentence networks of 75 units. The system was trained for the first 100 epochs with 0.1 learning rate, then 25 epochs with 0.05 and another 25 epochs with 0.025. The training process took about one hour on a Cray X-MP/48.

3.2 Results

The performance of the system was tested with the same set of sentences as used in the training. Table 3 show the performance figures for each network. In the output text, the system gets approximately 97% of the words (and punctuation) correct.

Even when the networks are connected in a chain (output of one network feeding the input of the next), the errors do not cumulate in the chain. The noise in the input is efficiently filtered out, and each network performs approximately at the same level. The figures for the sentence parser are somewhat lower because it generates expectations for the second and third acts. For some one and two act sentences these patterns remain active after the whole sentence has been read in. For example, after reading `The woman blamed the man` the network generates an expectation for a relative clause attached to man. The act generator network learns not to output the expectations, but they are counted as errors in the performance figures for the sentence generator.

4 Discussion

It is interesting to speculate how the model would map onto human sentence processing. The act parser network models the lowest level of processing. As each act fragment is read in, a surface semantic interpretation of it is immediately formed in terms of case roles. Each act fragment is parsed independently from others. A higher-level process (the sentence parser) keeps track of the recursive relations of the act fragments and combines them into complete representations. It also ties the different acts together by determining the referents of the relative pronouns.

The acts are stored in the memory as separate facts, without explicit high-level structure. The structure is represented in the facts themself, e.g. two acts have the same agent, the agent of one act is the patient of another etc. Sentences with relative clauses can be produced from this unstructured internal representation.

In other words, the *recursive structure is a property of the language, not the information itself.* Internally, the information can be represented in a parallel, canonical form, which makes all information directly accessible. In communication through narrow channels, i.e. in language, it is necessary to transform the knowledge into a sequential form [12]. Parallel dependencies in the knowledge are then coded with recursion.

Generating output is seen as a task separate from parsing. Sentence generation is performed by a different module and learned separately. The same module can learn to paraphrase the same internal representation in different ways, e.g. as a single sentence consisting of relative clauses, or as a sequence of several simple sentences. What actually gets generated depends on the connection weights of this module.

It would be possible to add a higher-level decision-making network to the system, which controls the connection weight values in the sentence generator network through multiplicative connections [14]. A decision about the style, detail etc. of the paraphrase would be made by this module, and its output would assign the appropriate function to the sentence generator.

The model exhibits certain features of human performance. As recursion gets deeper, the sentence networks have to keep more information in their sequence memories, and the performance degrades. Moreover, tail recursion (e.g. `The woman blamed the man, who hit the girl, who blamed the boy`) is easier than relative clauses in the middle of the sentence (e.g. `The woman, who the boy, who the girl blamed, hit, blamed the man`), because the latter case involves more steps in the sequence, taxing the memory capacity more. Note that in symbolic modeling, the depth or the type of the recursion makes absolutely no difference.

The scale-up prospects of the architecture seem fairly good. The simple data used in the experiments did not come close to exhausting the processing power of the system. Larger vocabulary, more case roles and sentences consisting with more acts could well be processed. It seems possible to represent a wide range of acts by their case-role assignments. Complex attributes, such as PPs, can be represented as additional relative clauses (e.g. `The man with the hat...` → `The man, who has the hat...`).

Currently, the system depends on commas to separate the clause fragments. This is not a very serious limitation, as segmenting could be based other markers such as the relative pronouns. A more fundamental limitation, characteristic to PDP systems in general, is that the system needs to be trained with a good statistical sample of the input/output space. It does not have an abstract representation of the clause structure, and it cannot generalize into sentence structures it has not seen before.

As sentences become more complex, a mechanism for maintaining unique identities for the words is needed. For example, in representing `The man, who helped the boy, blamed the man, who hit the girl` it is crucial to indicate that the man-who-helped is the same as the man-who-blamed, but different from the man-who-hit. A possible technique for doing this has been proposed in [12]. The representation of the word could consist of two parts: the content part, which is developed by FGREP and codes the processing properties of the word, and an ID part, which is unique for each separate instance of the word. The ID approximates sensory grounding of the word, and allows us to tag the different instances and keep them separate.

5 Conclusion

Dividing the task of parsing and generating sentences with complex clause structure into hierarchical subtasks makes the task tractable with distributed neural networks. The scale-up prospects of the approach into larger vocabulary and more complex sentences seem fairly good. The main drawback is that the system does not develop an abstract representation of recursive structures, but must be exposed to examples of all possibilities. The content of the sentences can be represented internally in canonical form as a collection of simple acts, without explicit structure. The knowledge for generating different linguistic expressions of the same content resides in the generating modules.

References

[1] E. Charniak. A neat theory of marker passing. In *Proceedings of AAAI-86*, Kaufmann, 1986.

[2] G. W. Cottrell and S. L. Small. A connectionist scheme for modelling word sense disambiguation. *Cognition and Brain Theory*, 6(1):89–120, 1983.

[3] M. G. Dyer. *In-Depth Understanding*. MIT Press, 1983.

[4] J. L. Elman. *Finding Structure in Time*. Technical Report 8801, Center for Research in Language, UCSD, 1988.

[5] R. H. Granger, K. P. Eiselt, and J. K. Holbrook. Parsing with parallelism. In Kolodner and Riesbeck, *Experience, Memory and Reasoning*, LEA, 1986.

[6] G. E. Hinton. Representing part-whole hierarchies in connectionist networks. In *Proceedings of CogSci-88*, LEA, 1988.

[7] T. E. Lange and M. G. Dyer. High-level inferencing in a connectionist network. *Connection Science*, 1(2), 1989.

[8] J. L. McClelland and A. H. Kawamoto. Mechanisms of sentence processing. In McClelland and Rumelhart, *Parallel Distributed Processing*, MIT Press, 1986.

[9] M. C. McCord. Using slots and modifiers in logic grammars for natural language. *Artificial Intelligence*, 18:327–367, 1982.

[10] R. Miikkulainen and M. G. Dyer. Encoding input/output representations in connectionist cognitive systems. In Touretzky, Hinton, and Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, Kaufmann, 1989.

[11] R. Miikkulainen and M. G. Dyer. A modular neural network architecture for sequential paraphrasing of script-based stories. In *Proceedings of IJCNN-89*, 1989.

[12] R. Miikkulainen and M. G. Dyer. *Natural Language Processing with Modular Neural Networks and Distributed Lexicon*. Technical Report UCLA-AI-90-02, Computer Science Department, UCLA, 1990.

[13] F. C. N. Pereira and D. H. Warren. Definite clause grammars for language analysis. *Artificial Intelligence*, 13:231–278, 1980.

[14] J. Pollack. Cascaded back-propagation on dynamic connectionist networks. In *Proceedings of CogSci-87*, LEA, 1987.

[15] J. Pollack. Recursive auto-associative memory. In *Proceedings of CogSci-88*, LEA, 1988.

[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In Rumelhart and McClelland, *Parallel Distributed Processing*, MIT Press, 1986.

[17] R. Schank and R. Abelson. *Scripts, Plans, Goals, and Understanding*. LEA, 1977.

[18] M. F. St. John and J. L. McClelland. Learning and applying contextual constraints in sentence comprehension. In *Proceedings of CogSci-88*, LEA, 1988.

[19] R. A. Sumida, M. G. Dyer, and M. Flowers. Integrating marker passing and connectionism for handling conceptual and structural ambiguities. In *Proceedings of CogSci-88*, LEA, 1988.

[20] D. L. Waltz and J. B. Pollack. Massively parallel parsing. *Cognitive Science*, 9:51–74, 1985.