Natural Language Processing With Modular PDP Networks and Distributed Lexicon [*][†][‡]

Risto Miikkulainen and Michael G. Dyer
University of California, Los Angeles

Abstract

An approach to connectionist natural language processing is proposed, which is based on hierarchically organized modular Parallel Distributed Processing (PDP) networks and a central lexicon of distributed input/output representations. The modules communicate using these representations, which are global and publicly available in the system. The representations are developed automatically by all networks while they are learning their processing tasks. The resulting representations reflect the regularities in the subtasks, which facilitates robust processing in the face of noise and damage, supports improved generalization, and provides expectations about possible contexts. The lexicon can be extended by cloning new instances of the items, that is, by generating a number of items with known processing properties and distinct identities. This technique combinatorially increases the processing power of the system. The recurrent FGREP module, together with a central lexicon, is used as a basic building block in modeling higher level natural language tasks. A single module is used to form case-role representations of sentences from word-by-word sequential natural language input. A hierarchical organization of four recurrent FGREP modules (the DISPAR system) is trained to produce fully expanded paraphrases of script-based stories, where unmentioned events and role fillers are inferred.

1 Introduction

Parallel distributed processing (PDP) models have recently emerged as an alternative to symbolic modeling in cognitive science. Their major appeal is that the processing knowledge can be extracted automatically from examples. The same architecture can learn to process a wide variety of inputs and take advantage of the implicit statistical regularities in the data, without having to be specifically programmed with particular data in mind (McClelland et al., 1986). The gradual evolution of the system performance as it is learning often resembles human learning in the same task (Rumelhart and McClelland, 1987; Sejnowski and Rosenberg, 1987).

Such PDP models typically have very little internal structure or architectural complexity. They produce the statistically most likely answer given the input conditions, in a process which is opaque to the external observer. This approach is well suited for modeling isolated low-level tasks, such as learning past tense forms of verbs or pronunciation of words. However, modeling higher level cognitive tasks with homogeneous networks has been unfeasible, for three reasons:

---

[*] This research was supported in part by a grant from the Initial Teaching Alphabet (ITA) Foundation. The first author was also supported in part by grants from the Academy of Finland, the Finnish Science Academy, The Emil Aaltonen Foundation, and the Foundation for Advancement of Technology (Finland). The simulations were carried out on Cray Y–MP8/864 at the San Diego Supercomputer Center, and on equipment donated to UCLA by Hewlett Packard. Special thanks go to James L. McClelland, David Touretzky and an anonymous reviewer for thoughtful comments on earlier drafts of this article.

[†] 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.

[‡] To appear in *Cognitive Science*, 15(3): 343–399, 1991.

1. Higher level tasks usually require composite subtasks. They consist of several distinct subprocesses, such as parsing language, generating language, memory storage, memory retrieval, and reasoning, which cannot be performed in a single pattern transformation. Complex behavior requires bringing together several different kinds of knowledge and processing, which is not possible without structure (Feldman, 1989; Simon, 1981).
2. The required network size, the number of training examples and the training time becomes intractable as the size of the problem grows, especially in sequential processing (Harris and Elman, 1989; Servan-Schreiber et al., 1989).
3. There is no way to evaluate what the entire system is doing, for example, what knowledge it is acquiring and applying, unless each module processes meaningful internal representations that can be interpreted by an external observer and by other modules in the system.

A plausible approach for higher level cognitive modeling, therefore, is to construct the architecture from several interacting modules, which work together to produce the higher level behavior (Minsky, 1985). Central issues to be addressed in this approach are:

1. How the overall task should be broken into modules and how the modules should be organized;
2. How PDP subnetworks should be designed so that they can serve as modular building blocks;
3. How communication among such building blocks can be established; and
4. How to make use of the modular structure in training the system.

Our discussion begins with the communication issue (3). Any complex architecture will need to have a common set of terms, serving the function of a "symbol table" for intercommunication. In a large system consisting of many modules, with many communicating pairs, the most efficient way to establish this function is through a global vocabulary, a central lexicon. The modules communicate using terms from this global symbol table, instead of having a separate set of terms for each communication channel. Each module can then be trained separately and in parallel, as long as they are trained with compatible input/output (I/O data.

In a high-level PDP model, communication (i.e., input and output of each module) can take place using distributed representations. This is a major advantage of the PDP approach in general. Distributed representations can reflect the meanings of the items they stand for. Similar items have similar representations, which results in several interesting processing enhancements (McClelland et al., 1986; van Gelder, 1989). Information is inherently content addressable, and fuzzy categories can be naturally represented. Performance of a PDP system degrades gracefully in less than perfect conditions, such as with overload, damage, and noisy or incomplete input. PDP networks also spontaneously generalize their processing knowledge to previously unseen situations.

In backpropagation networks with hidden layers, the network automatically develops internal distributed representations for the I/O items as a side effect of learning the processing task (Elman, 1990; Hinton, 1986; Miikkulainen and Dyer, 1987; Rumelhart et al., 1986b). These representations reflect the regularities of the task and data, extracted without external supervision, and provide an excellent basis for generalization. We present a mechanism for forming distributed representations, FGREP, where *the internal representations discovered by the network are made public in a global lexicon*, so that they can be used for communication in a large modular system.

Our discussion concentrates on natural language processing (NLP) tasks, where the lexicon consists of distributed representations of words. The FGREP mechanism is first introduced in the context of sentence processing, in the specific task of assigning roles to sentence constituents. The method is then extended to sequential input, where word representations are developed in the task of mapping a sequence of input words into the case-role representation of the sentence.

The input and output of a building block must be self-contained and publicly interpretable, so that other modules can be trained to use the same data (Issue 2). Each module has to represent its processing knowledge explicitly in its output. The FGREP architecture was designed according to this principle, and therefore its application to sentence processing differs from other recent architectures (Elman, 1990; St. John and McClelland, 1990). We are interested in sentence processing as a subtask in language understanding, and we are concerned with integrating sentence processing into the larger context. The sentence case-role representation is publicly interpretable and can be used as input to other modules.

We will address Issues 1 and 4 by demonstrating how a significantly more complex system, DISPAR, can be built from hierarchically organized recurrent FGREP modules, together with a central lexicon of words. Hierarchical organization with a sequential interface is well suited for language-processing tasks, and it also turns out to be a very efficient way to reduce complexity. The FGREP modules can be trained separately and in parallel, making good use of the modular architecture. Trained with compatible I/O data and developing the same lexicon, they perform well together when connected. The DISPAR system learns to produce fully expanded paraphrases of script-based input stories, where unmentioned events are inferred, and unspecified fillers are inferred.

The modular FGREP architecture does not aim at direct biological plausibility. The fact that processing emerges collectively from a large number of simple units operating in parallel, and is based on distributed representations, is obviously neurally inspired, but we are not trying to model the physical structures of the brain. In this work, neural networks are seen as a class of statistical machines, which have several useful properties for natural language processing. A major motivation for our work is to give a better account for high-level phenomena, based on these properties.

Several interesting high-level phenomena emerge from the FGREP/DISPAR architecture. The FGREP representations are found to encode the implicit categorization of words, to form a basis for robust processing and to facilitate generalization. Sequential expectations arise automatically from the representations in the recurrent FGREP module, and the expectations automatically implement script-based inference in DISPAR. In other words, script-based inferencing is grounded in the statistical regularities in the input examples, an issue which has not been addressed by symbolic script-processing models such as SAM (Cullingford, 1978).

2 Methods for forming distributed representations

Sentence case-role assignment is an example of a cognitive task that is well suited for modeling with connectionist systems. Case-role assignment requires taking into account all the positional, contextual, and semantic constraints simultaneously, which is what the connectionist systems are particularly good at. An important issue is how the input and output to such systems should be encoded.

In the distributed approach, different I/O items are represented as different patterns of activity over the same set of units. One approach for forming these patterns is *semantic feature encoding*, used, for example, by McClelland and Kawamoto (1986) in the case-role assignment task (see, also, Hinton, 1981). Each concept is classified along a predetermined set of dimensions such as human–nonhuman, soft–hard, and male–female. Each feature is assigned a processing unit (or a group of units, e.g. one for each value), and the classification becomes a pattern of activity over an assembly of units.

This kind of representation is meaningful by itself. It is possible to extract information just by examining the representation, without having to train a network to interpret it. Several different systems can process the same representations and communicate using them.

On the other hand, such patterns must be pre-encoded and they remain fixed. Performance cannot be optimized by adapting the representations to the actual task and data. Because all concepts must be classified along the same dimensions, the number of dimensions becomes very large, and many of them are irrelevant to the particular concept (e.g., gender of `rock`). Deciding what dimensions are necessary and useful is a hard problem (van Gelder, 1989). There is also the epistemological question of whether the process of deciding what dimensions to use is justifiable or not. Hand-coded representations are always more or less ad hoc and biased. In some cases it is possible to make the task trivial by a clever encoding of the input representations.

Developing internal representations in hidden layers of a backpropagation network avoids these problems. Hinton's (1986) family-tree network is a good example. That network consists of input, output, and three hidden layers. The input and output layers are localist: Exactly one unit is dedicated to each item. The hidden layers next to the input and output layers contain considerably fewer units, which forces these layers to form compressed distributed activity patterns for the input and output items. Developing these patterns occurs as an essential part of learning the processing task, and they end up reflecting the regularities of the task (Hinton, 1986).

Another variant of the same approach was proposed by Elman (1989, 1990). A simple recurrent network is trained to predict the next word in the input word sequence. The hidden layer of the network develops structured representations for the words based on how the words occur in the sequences.

This approach does not address the issue of *encoding I/O representations*. Hinton's and Elman's systems do not deal with the representations per se; they develop as a side effect of modifying the weights to improve performance in the task. The patterns are not available outside the network, and they are not used in communicating with the network. Moreover, each item has a different representation at each hidden layer. For example, the penultimate layers of the family-tree network develop one representation for input and a different one for output. The hidden layer patterns are *local, internal processing aids* more than I/O representations which can be used in a larger environment.

In the *FGREP* approach (Miikkulainen and Dyer, 1987, 1988, 1989a), the representations are also developed automatically while the network is learning the processing task, by making use of the backpropagation error signal. However, the representations are *global* input and output to the network and are stored in an external network (a lexicon), which guarantees unambiguity and makes communication using these representations possible.

3 FGREP: Forming Global Representations with Extended back-Propagation

3.1 Basic FGREP architecture

The FGREP mechanism is based on a three-layer backward error-propagation network (Figure 1). The network learns the processing task by adapting the connection weights according to the standard backpropagation equations (Rumelhart et al., 1986b, pp. 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 by concatenating the current lexicon entries of the input items; likewise, it forms the corresponding target pattern by concatenating the lexicon entries of the target items. Thus, each item is represented by the same pattern in the input and in the output of the backpropagation network.

The representation–formation process begins with a random lexicon containing no pre-encoded

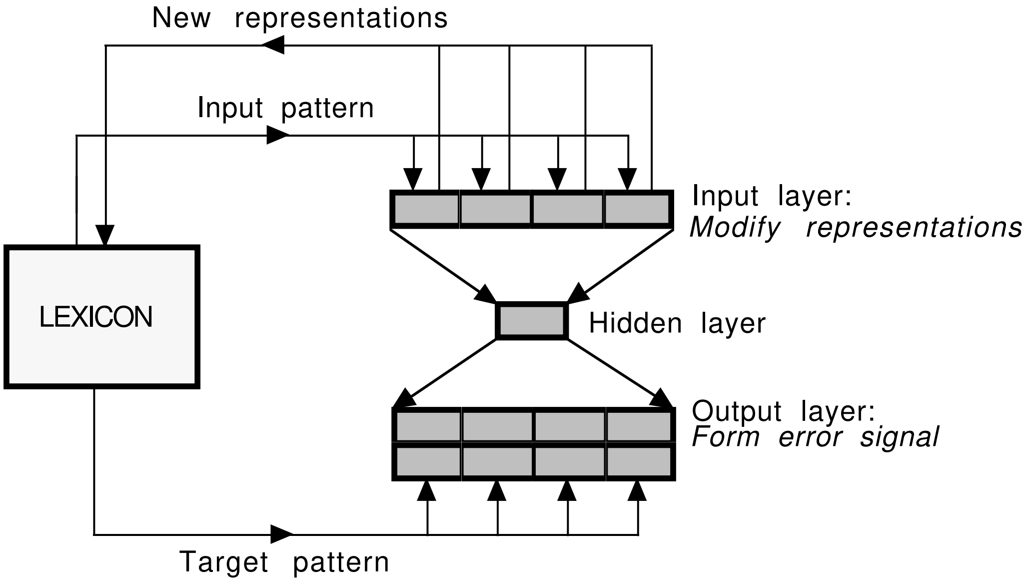

Figure 1: **Basic FGREP architecture.** The system consists of a three-layer backpropagation network and an external, global lexicon which contains the I/O representations. Each input and output assembly (grey rectangle in the figure) holds a single word representation, taken from the lexicon. In the figure, four words are concatenated in the input and output layers. 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.

information. During the course of learning, the representations adapt to reflect the implicit regularities of the task. It turns out that single components in the resulting representations do not necessarily have a clear interpretation. The representation does not implement a classification of the item along identifiable features (as in semantic feature encoding). In the most general case, the representations are simply profiles of continuous activity values over a set of processing units. The representation pattern as a whole is meaningful, and can be claimed to code the meaning of the item. The representations for items used in similar ways become similar.

3.2 Extending backpropagation to the representations

Standard backpropagation produces an error signal for each hidden layer unit. By propagating the signal one layer farther to the input layer, representations can be changed as if they were an extra layer of weights.

In a sense, the representation of an item serves as input activation to the input layer. The activation of an input unit is identical to the corresponding component in the representation. In this analogy, the activation function is the identity function and its derivative is one. The error signal can be computed for each input unit as a simple case of the general error signal equation (Rumelhart et al., 1986b, Eq.14, pp. 326):

$$\delta_{1i} = \sum_j \delta_{2j} w_{1ij}. \tag{1}$$

where δ_{xy} stands for the error signal for unit y in layer x, and w_{1ij} is the weight between unit i in the input layer and unit j in the first hidden layer. The representations are now changed according to the error signal:

$$\Delta r_{ci} = \eta \delta_{1i}, \tag{2}$$

where r_{ci} is the representation component i of item c, δ_{1i} is the error signal of the corresponding input layer unit and η is the learning rate. This is the standard adjustment in backpropagation; only instead of weights, we are now adjusting representations. In other words, *representation learning is implemented as an extension of the backpropagation algorithm.* The weight values are unlimited, but the representation values must be limited between the maximum and minimum activation values of the units. The new value for the representation component i of item c is obtained as

$$r_{ci}(t+1) = max[o_l,\ min[o_u,\ r_{ci}(t)\ + \Delta r_{ci}]], \tag{3}$$

where o_l is the lower limit and o_u is the upper limit for unit activation.

Note that the backpropagation algorithm "sees" the representations simply as an extra layer of weights. Because the representations adapt according to the error signal, there is reason to believe that *the resulting representations will effectively code properties of the input elements that are most crucial to the task.*

3.3 Reactive training environment

The process differs from ordinary backpropagation in that *both the input and the target patterns are changing.* An input pattern is formed by drawing the current representations of the input items from the lexicon and loading them into the input assemblies (Figure 1). The activity is propagated through the network to the output layer, where the error signal is formed by comparing the output pattern to the target pattern, which is also formed by selecting representations from the lexicon. The error signal is propagated back to the input layer, changing both weights and the input item representations along the way. Next time the *same input* occurs, the output will be closer to the *same target pattern.* The modified input representations are now put back into the lexicon, replacing the old ones and thereby *changing the next target pattern for the same input.* In other words, the shape of the error surface is changing at each step, and backpropagation is shooting at a moving target in a reactive training environment.

It turns out that as long as the changes made in the process are small, the process converges nevertheless. Learning time appears to be about the same as in the ordinary case. The modifiable input patterns form an additional set of parameters which the system can use to learn the task. Changes in the error surface are a form of noise (a noisy error signal), which backpropagation in general tolerates very well.

It is conceivable that the representations might converge into a trivial solution, where they are all identical. Such a situation has never occurred in our experiments, and our sense is that it is very unlikely to occur when the representations are initially random. It seems that starting from a random point in the representation-set space, the process is much more likely to converge into a nontrivial stable solution. Apparently, these solutions are more numerous and more evenly distributed in the solution space than the trivial solutions. Moreover, if some of the representations are fixed (e.g., the "blank" representation at all-0, and the "don't care" representation at all-0.5), all-identical representation sets are excluded altogether.

4 An example task: Assigning case roles to sentence constituents

Case-role representation of sentences is based on the theory of thematic case roles (Fillmore, 1968), adapted for computer modeling in conceptual dependency theory (Schank and Abelson, 1977; Schank and Riesbeck, 1981). In the basic version of the case-role assignment task, the syntactic structure of the sentence is given and consists of, for example, the subject, verb, object, and a with-clause. The task is to decide which constituents play the roles of agent, patient, instrument, and patient modifier in the act. This task requires forming a shallow semantic interpretation of the sentence.

For example, in `The ball hit the girl with the dog`, the subject `ball` is the instrument of the `hit` act, the object `girl` is the patient, the with-clause `dog` is a modifier of the patient, and the agent of the act is unknown. Role assignment is context dependent: in `The ball moved`, the same subject `ball` is taken to be the patient. Assignment also depends on the semantic properties of the word.[1] In `The man ate the pasta with cheese`, the with-clause modifies the patient; but in `The man ate the pasta with a fork`, the with-clause is the instrument. In yet other cases, the assignment must remain ambiguous. In `The boy hit the girl with the ball`, there is no way of telling whether `ball` is an instrument of `hit` or a modifier of `girl`.

McClelland and Kawamoto (1986) described a system that learns to assign case roles to sentence constituents. FGREP was tested in the same task, partly because it provides a convenient comparison to a system based on semantic feature encoding. The task was restricted to a small repertoire of sentences studied in the original experiment. These sentences were generated from the sentence templates and noun categories listed in Appendix 13. Note that the templates and categories are not visible to the system: They are only manifest in the combinations of words that occur in the input sentences. To do the case-role assignment properly, the system has to figure out the underlying relations and code them into the representations.

In this particular task, the target pattern is made up from the input sentence constituents (Figure 2). This structure is by no means necessary for learning the representations. The required output could be anything and the FGREP method would work the same. A discriminatory or “pigeonholing” task, such as case-role assignment, is actually harder than a general I/O mapping task because of the reactive training effect.

5 Properties of FGREP-representations

5.1 Simulations

FGREP learning is fairly insensitive to the simulation parameters and the system-configuration parameters. The online version of backpropagation (weights and representations are updated after each presentation instead of after each epoch) without momentum turns out to be the most efficient training method. Best results are obtained by presenting the sentences within each epoch in different random order, and by gradually reducing the learning rate. The results reported in the following three sections are from a run with the learning rate $\eta = 0.1$ for the first 200 epochs, 0.05 until 500, and 0.025 until 600.

The number of units in the representation and the number of hidden units are not crucial either: As few as 5, and as many as 100, were tried. If more hidden units are used, the task performance and damage resistance improve slightly, and learning in general is faster. Decreasing the number of hidden units, on the other hand, places more pressure on the representations, and they become more descriptive faster. In general, best results are obtained when the number of hidden units is about half the number of units in the input layer. In the example simulation, 12 units were used for each word representation (i.e., 48 input units total) and 25 for the hidden layer. The representation components were initially uniformly distributed in the interval [0,1] and the connection weights within [-1,1]. Figure 2 shows a snapshot of the simulation after a real-time display on an HP 9000/350 workstation.

---

[1] The term “word” and labels such as `ball` are used here in the conceptual sense to refer to the concept, not to the lexical item. It is assumed that there exists a one-to-one mapping between lexical items and concepts and the lexicon automatically performs this mapping, so that the described system only needs to deal with concepts (see Miikkulainen, 1990a, 1990b). Ways to deal with ambiguity are discussed in Section 11.2.

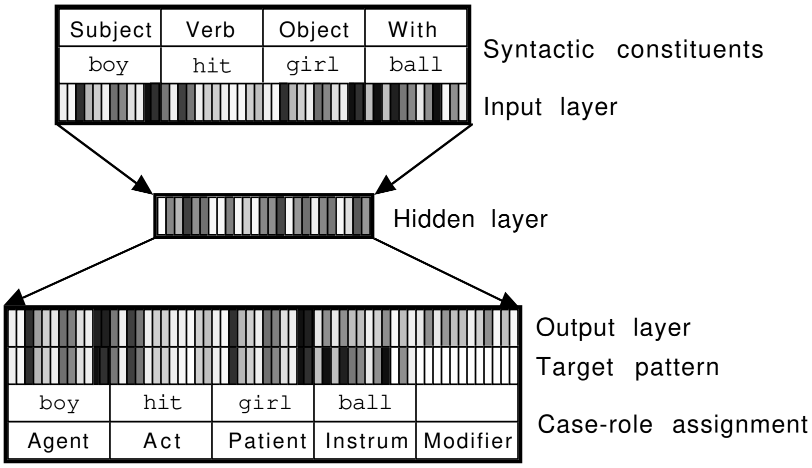

Figure 2: **Snapshot of basic FGREP simulation.** The input and output layers of the network are divided into assemblies, each holding one word representation at a time. Each unit in an input assembly is set to the activity value of the corresponding component in the lexicon entry. The input layer is fully connected to the hidden layer and the hidden layer to the output layer. Connection weights are omitted from the figure. If the network has successfully learned the task, each output assembly forms an activity pattern identical to the lexicon representation of the word filling that role. The correct role assignment is shown at the bottom of the display. This pattern forms the output target for the network. Grey-scale values from white to black are used in the figure to code the unit activities, which vary within the range [0,1].

The next four sections concentrate on a baseline simulation, where 1,439 of the 1,475 sentences generated by the templates were used for training, and 38 were reserved for testing. Generalization as a function of the training set size is discussed in Section 5.6.

5.2 Final representations

Starting from random representations, the similarity of the nouns belonging to the same category first increases rapidly until the changes begin to cancel out. Categorization is in a dynamic equilibrium (i.e., representations change but categorization does not improve) while the task performance improves. The decreasing error signal and learning rate eventually allow fine tuning the representations and they converge into a stable, descriptive categorization. Figure 3 displays the final noun representations, organized according to the categories. With different initial configurations, the final set of representations would look different, but the overall similarities of the representations and the performance of the system would be about the same.

Some words belong exactly to the same categories and, consequently, occur exactly in the same contexts. They are indistinguishable in the data and their representations become identical.[2] There are eight such groups of words in the data: {`man`, `woman`, `boy`, `girl`}, {`fork`, `spoon`}, {`wolf`, `lion`}, {`plate`, `window`}, {`ball`, `hatchet`, `hammer`}, {`paperwt`, `rock`}, {`desk`, `curtain`}, and {`cheese`, `pasta`, `carrot`}. If there is at least one difference in the usage of two nouns, their representations tend to become different. The discriminating input modifies one of the representations while the other one remains the same.

Because each noun belongs to several categories, its representation can be seen as evolving from the competition among the categories. This is clearest on the part of the ambiguous nouns `chicken`

---

[2] Random ordering of training examples together with noninfinitesimal learning rate introduces a certain amount of noise into the representations. In practice, two representations are never *exactly* identical, and minute but theoretically significant differences are sometimes obscured by noise. Fortunately, most of the interesting similarities and differences are obvious above noise level.

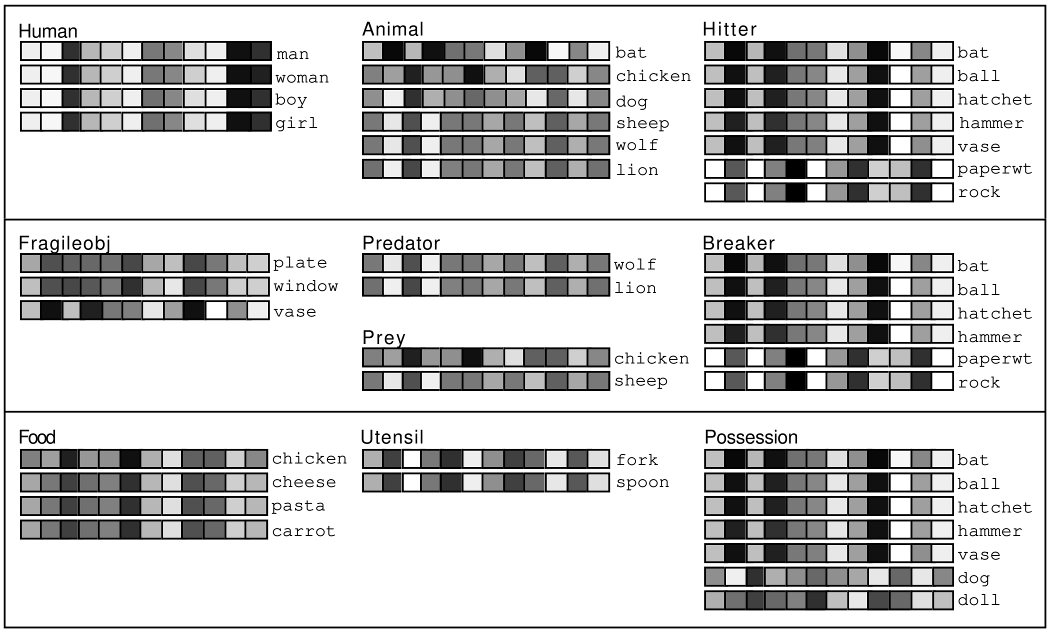

Figure 3: **Final representations.** The representations for the synonymous words {`man`, `woman`, `boy`, `girl`}, {`fork`, `spoon`}, {`wolf`, `lion`}, {`plate`, `window`}, {`ball`, `hatchet`, `hammer`}, {`paperwt`, `rock`} and {`cheese`, `pasta`, `carrot`} have become almost identical.

and `bat`, which are both animals, but `chicken` is also food and `bat` is a hitter. The representation is a combination of both, weighted by the number of occurrences of each meaning (there are more plausible ways to deal with ambiguous words; see Section 11.2). On the other hand, the fact that there is a common element in two categories tends to make all representations of the two categories more similar. The properties of one word are generalized, to a degree, to the whole class.

Note that the categorization of a word in Figure 3 is formed outside the system and is independent of the task, other categories, and other words. The system itself is not attempting categorization; it is forming the most efficient representation of each word for a particular task. Interestingly, hierarchical merge clustering algorithm (Hartigan, 1975) finds optimal clusters quite similar to the noun categories (Figure 4).

Inspection of the representations in Figure 3 suggests that a single unit does not play a crucial role in the classification of items. The fact that a word belongs to a certain category is indicated by the *activity profile as a whole*, instead of particular units being on or off. The representations are also extremely *holographic*. The whole categorization is visible even in the values of a single unit (Figure 5).

Each unit provides a unique perspective to the words, by coding slightly different properties of the word space. Combining the values of two units thus provides an even more descriptive categorization, and the complete representation can be seen as a combination of 12 slightly different viewpoints into the word space.

To obtain insight into this combined categorization, we first have to map the 12-dimensional representation vectors into two dimensions. One way to do this is Kohonen's self-organizing feature mapping (Kohonen, 1984). This method is known to map clusters in the input space to clusters in the output space. The map is topological, that is, the distances in the map are not comparable (more dense regions are magnified), but the topological relations of the input space are preserved.

The feature map shows the same clusters that were used in generating the input (Figure 6). The ambiguous nouns and nouns belonging to several categories are mapped between the categories:

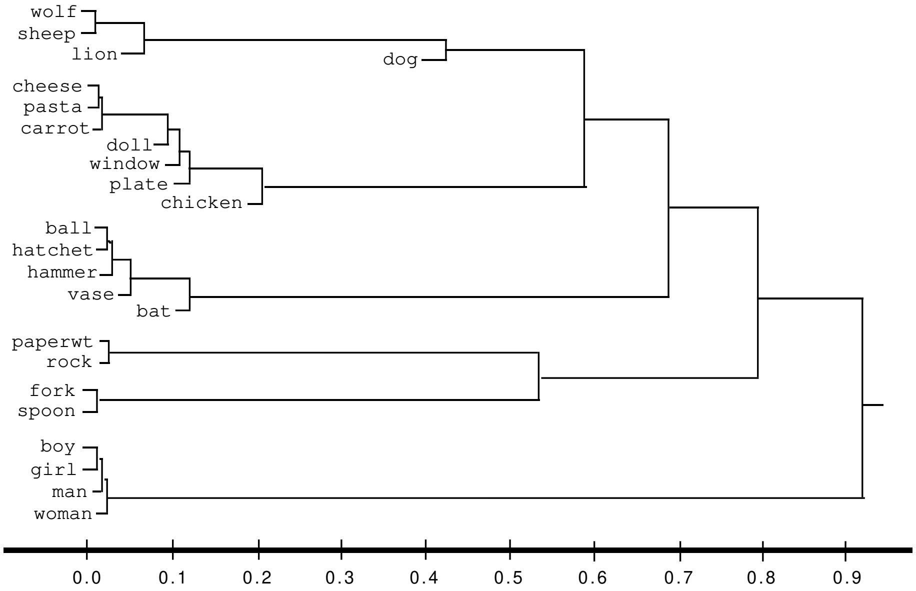

Figure 4: **Merge clustering the representations.** Step by step, the clusters with the shortest single linkage distance were merged. Distance is indicated on the horizontal axis. There are six prominent clusters with very small distances between items: animals, humans, utensils, two different types of hitters and a combination of foods and fragile objects. Ambiguous words (`chicken, bat`) and words with an unusual use (e.g., `dog`, which is a possession but not a hitter) do not fit very well into any category, and they are merged later in the process. Eventually the clusters are merged together, but only by bringing together very different representations.

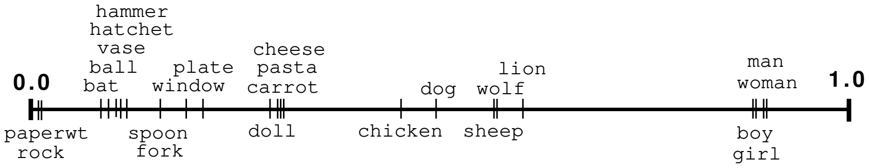

Figure 5: **Categorization by unit 11.** The words are placed on a continuous line [0,1] according to the value of the last unit in their representations (the units are numbered 0 – 11).

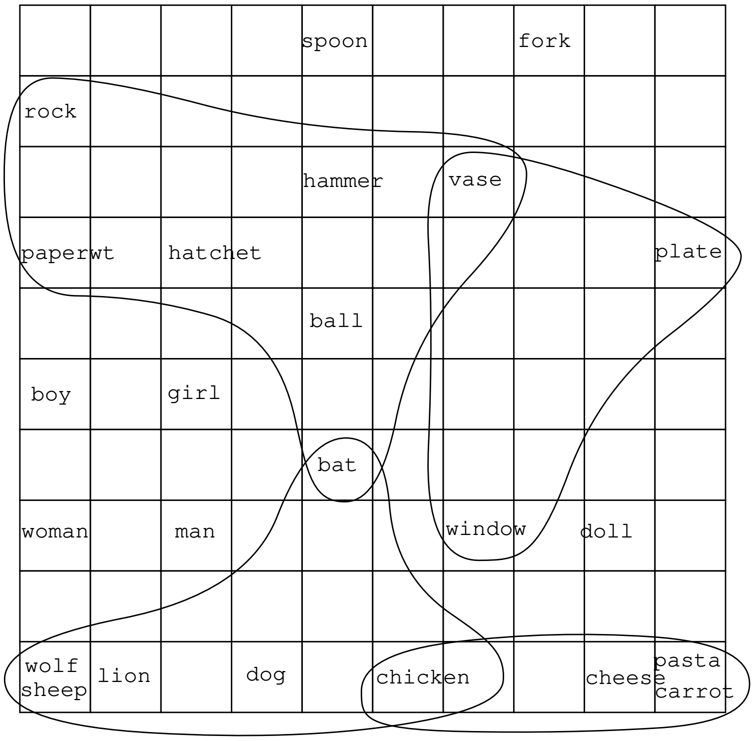

Figure 6: **2-D Kohonen-map of the representations.** Labels indicate the maximally responding unit in the 10 × 10 feature map network for each representation vector. The map was formed in 15,000 epochs, where the neighborhood radius was decreased from 4 to 1 and the learning rate from 0.5 to 0.05 during the first 1,000 epochs, and to 0 during the remaining epochs.

`chicken` is mapped between animal and food, `bat` between animal and hitter, and also `vase` between fragile-obj and hitter.

It is very hard to name the properties the individual units are coding. Hinton (1986) was able to give semantic interpretation for some of the units in the hidden layer representation, although he pointed out that the system develops its own microfeatures, which may or may not correspond to ones that humans would use to characterize data. Similar results have been reported by Pollack (1988) and Sejnowski and Rosenberg (1987).

Our results suggest that with complex data, the microfeatures in the resulting representation are identifiable only accidentally. The network is trying to code several dimensions of variation (i.e., feature dimensions) into a lower dimensional representation, and it has to distribute the features over several units. As a result, each individual unit becomes sensitive to a *combination of several features* which is very unlikely to exactly match an established term, although partial matches are possible. For example, unit 11 seems to separate animate objects, but the separation is not clear-cut and has a lot of finer structure (Figure 5). The network does not lose the information about animateness if this unit becomes defective (Section 5.4), so it is incorrect to say that the unit codes the "animate feature" of the input data.

The holographic property does not mean that the representations are uninterpretable. Instead of labeling single units, interpretation requires more sophisticated techniques that look at complete representation vectors, such as cluster analysis and feature mappings, or the use of functional criteria (van Gelder, 1989). It might also be possible to rotate the representation vector (e.g. through principal component analysis) so that some of the representation dimensions coincide with the human-preferred feature dimensions, and subsequently label the components of the rotated vector.

Template	Input				$E_i < .15$	E_{avg}
1.	man	ate			88	.067
1.	girl	ate			88	.066
2.	woman	ate	cheese		100	.018
2.	woman	ate	pasta		100	.018
3.	woman	ate	chicken	pasta	100	.015
3.	man	ate	pasta	chicken	100	.021
4.	girl	ate	pasta	spoon	100	.023
4.	boy	ate	chicken	fork	100	.025
5.	dog	ate			83	.068
5.	sheep	ate			82	.065
6.	lion	ate	chicken		97	.037
6.	lion	ate	sheep		98	.034
7.	woman	broke	window		100	.014
7.	boy	broke	plate		100	.014
8.	man	broke	window	bat	100	.016
8.	boy	broke	plate	hatchet	100	.014
9.	paperwt	broke	vase		100	.023
? 9.	bat	broke	plate		100	.028
?10.	bat	broke	window		68	.182
10.	wolf	broke	plate		100	.023
11.	vase	broke			93	.039
11.	window	broke			100	.024
12.	man	hit	pasta		100	.009
12.	girl	hit	boy		100	.023
?13.	man	hit	girl	hatchet	77	.093
?13.	man	hit	woman	hammer	77	.092
14.	woman	hit	bat	hammer	100	.008
14.	girl	hit	vase	bat	100	.011
15.	hatchet	hit	pasta		100	.008
15.	hammer	hit	vase		100	.014
16.	man	moved			93	.054
16.	woman	moved			93	.054
17.	woman	moved	plate		100	.010
17.	girl	moved	pasta		100	.010
?18.	bat	moved			80	.118
18.	dog	moved			87	.047
19.	doll	moved			100	.025
19.	desk	moved			100	.020
Average:					95	.038

Table 1: **Performance, familiar sentences.**

5.3 Performance in the task

The performance of the system in the role-assignment task was tested with two test sets. The first one consisted of two sentences from each template that had been used in the training, and the second one consisted of the test sentences the network had not seen before. Tables 1 and 2 show the results for each sentence. The leftmost entry in each row identifies the template that generated the sentence (referring to Table 5 in Appendix 13). The first number after each sentence indicates the percentage of output units whose values were within .15 of the correct output value (which ranged from 0 to 1). The second number indicates the average error per unit. Ambiguous sentences are marked with "?". The test sentence sets are identical to those used by McClelland and Kawamoto (1986).

The system learns the correct assignment for most sentences. Note that perfect performance is not possible with this data because some of the sentences are ambiguous. In these cases the system develops an intermediate output between the two possible interpretations, indicating a degree of confidence in the choices. One such case is presented in the snapshot of Figure 2. `Ball` can either be the instrument of `hit` or a possession of `girl`. In most similar occasions it is the instrument, and the network develops a slightly stronger representation in the instrument assembly. The system also performs poorly with the animal meaning of `bat`. Because a vast majority of the occurrences of `bat` are hitters, its pattern becomes more representative of hitter than animal.

Direct performance comparison with McClelland and Kawamoto's system is tricky because of different architectures and goals. Their system was based on binary units, where 96% of the output

Template	Input				$E_i < .15$	E_{avg}
1.	boy	ate			88	.066
1.	woman	ate			88	.066
2.	woman	ate	chicken		100	.018
2.	man	ate	chicken		100	.018
3.	woman	ate	chicken	carrot	100	.015
3.	boy	ate	carrot	pasta	100	.012
4.	man	ate	chicken	fork	100	.025
4.	woman	ate	carrot	fork	100	.023
5.	bat	ate			67	.186
5.	chicken	ate			68	.116
6.	wolf	ate	chicken		98	.036
6.	wolf	ate	sheep		98	.034
7.	girl	broke	plate		100	.014
7.	woman	broke	plate		100	.014
8.	man	broke	vase	ball	100	.016
8.	girl	broke	vase	hatchet	100	.016
9.	hammer	broke	vase		100	.018
9.	ball	broke	vase		100	.018
?10.	bat	broke	vase		68	.192
10.	dog	broke	plate		98	.032
11.	plate	broke			100	.026
11.	plate	broke			100	.026
12.	boy	hit	girl		100	.023
12.	girl	hit	carrot		100	.009
?13.	man	hit	boy	hammer	77	.092
13.	boy	hit	woman	doll	100	.026
14.	girl	hit	curtain	ball	100	.011
14.	girl	hit	spoon	rock	100	.007
15.	paperwt	hit	chicken		100	.014
15.	rock	hit	plate		100	.014
16.	boy	moved			93	.054
16.	girl	moved			93	.055
17.	man	moved	window		100	.011
17.	girl	moved	hammer		100	.011
18.	wolf	moved			82	.062
18.	sheep	moved			82	.062
19.	paperwt	moved			88	.056
19.	hatchet	moved			98	.024
Average:					94	.040

Table 2: **Performance, unfamiliar sentences.**

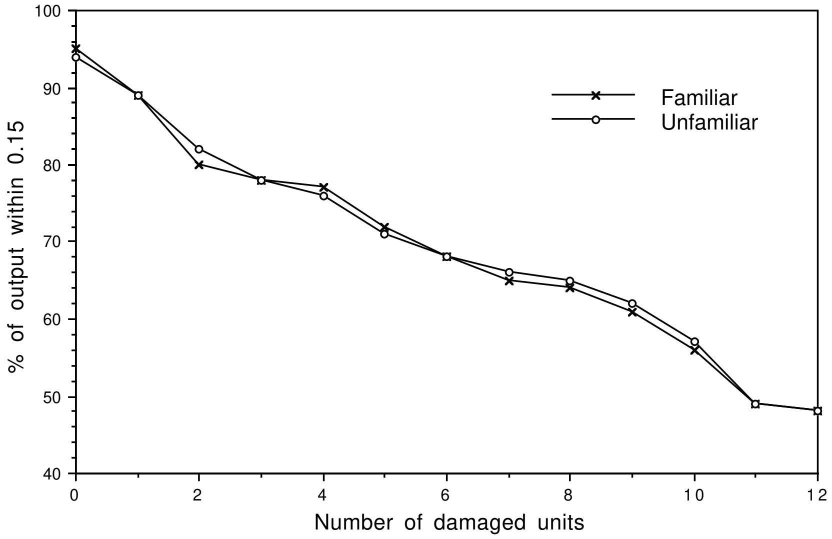

Figure 7: **Damage resistance.** The familiar and unfamiliar data sets are listed in Tables 1 and 2. The horizontal axis indicates the number of units eliminated from the 12-unit representation, and the vertical axis indicates the percentage of output units that were within .15 of the correct value.

units should be off on the average. Ninety-nine percent of all the output units were correct in their experiment, but only 85% of the units that should have been on were on. On the other hand, their system performed semantic enrichment of the representations at the output, as well as disambiguation of the different conceptual senses of `bat` and `chicken`, which had different representations in their data. These tasks were not studied in this FGREP experiment.

5.4 Damage resistance

The robustness of the representations against damage was tested by eliminating the last n units from each input assembly. These units were fixed at 0.5, the "don't care" value. Figure 7 shows the decline in performance as more and more units are eliminated. The decline is very gradual, and approximately linear. This is partly due to the general robustness of distributed networks, but also partly due to the fact that the representation is not coded into feature-specific units, but is distributed over all units in a holographic fashion. Eliminating a unit means removing one classification perspective, and these perspectives apparently are additive.

With a third of the input units removed, the system still gets 77% of the output within .15 of the correct value. The output patterns are still mostly recognizable at this level. Note also that even with all input units eliminated (i.e., without any information at the input layer), the system performs above chance level. Information about the input space distribution has been stored in the weights, and the network produces a best guess as an average of all possible outputs.

5.5 Creating expectations about possible contexts

The whole pattern in the input and target layers has an effect on how each input item is modified during backpropagation. The context of an input item can, therefore, be defined operationally as the whole I/O representation.

The representation for an item is determined by all the contexts where that item has been encountered. Consequently, the lexicon entry for that item is also *a representation of all these*

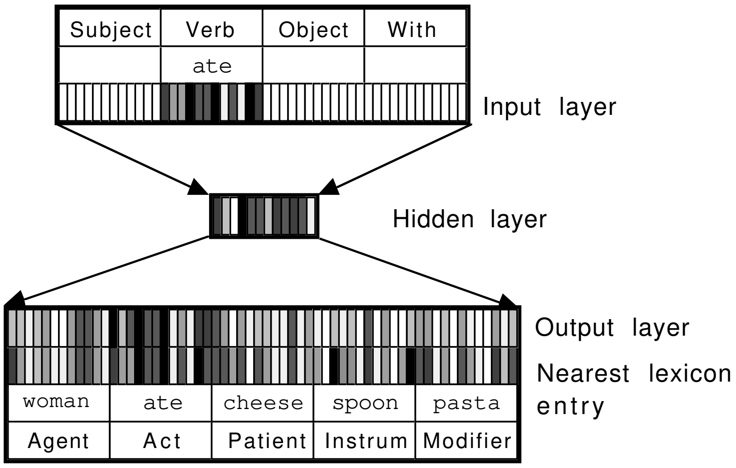

Figure 8: **Expectations embedded in the word `ate`.** The bottom layer shows the lexicon entries that are closest to the output generated by the network (Euclidian distance of normalized vectors). The effect is more pronounced when a small hidden layer is used. In this example, a network with 12 hidden units was trained for 50 epochs with a learning rate 0.1.

contexts. The more frequent the context, the stronger its trace in the representation. When a word is input into an FGREP module, a number of expectations about the context are automatically created at the output with different degrees of confidence. The expectations of different input words are combined to produce the total output pattern.

Expectations embedded in a single word are displayed when that word is used alone as the input (Figure 8). The resulting pattern at the output layer indicates the most likely context. As can be seen from Figure 8, the representations and the network have captured the fact that a likely agent for `ate` is human, patient is food, instrument is utensil, and that food can be eaten with another food.

Being able to create expectations automatically and cumulatively from the input representations turns out to be useful in building larger language-understanding systems. Such distributed expectations could replace the symbolic expectations traditionally used in natural language conceptual analyzers (e.g., Dyer, 1983).

5.6 Generalization

The term "generalization" commonly means processing inputs which the system has not seen before. In most cases this means extending the processing knowledge into new input patterns, which are different from all training patterns. Generalization in FGREP has a different character. The network *never has to extend its processing into very unfamiliar patterns, because generalization has already been done on the I/O representations.*

If an unfamiliar sentence is meaningful at all, its representation pattern is necessarily close to something the network has already seen. This is because FGREP develops similar representations for similarly behaving words. For example, the network has never seen `The man ate the chicken with a fork`, but its representation is very close to the familiar sentence, `The girl ate the pasta with a spoon`, because the representation for `girl` is equivalent to `man`, `fork` to `spoon`, and `chicken` is very much like `pasta`. In more general terms, the system can process the word `x` in situation S, because it knows how to process the word `y` in situation S, and the words `x` and `y` are used similarly in a large number of other situations.

If the pattern is far from familiar, the sentence cannot be meaningful in the microworld of the

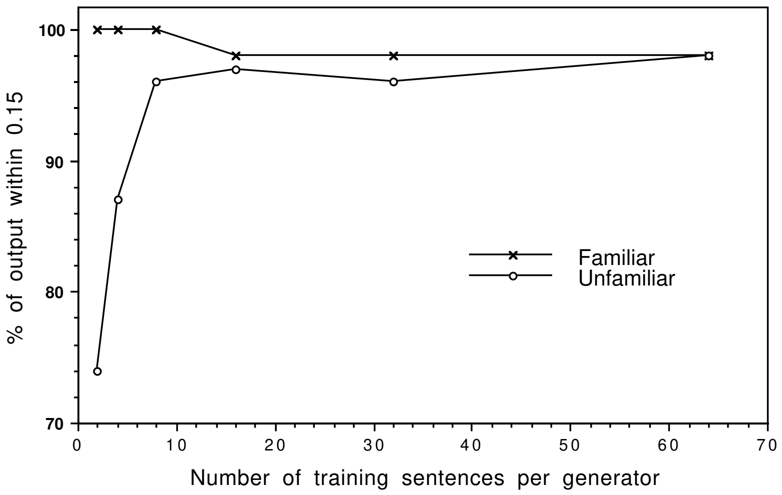

Figure 9: **Performance as a function of the training-set size.** The familiar and unfamiliar data sets are listed in Tables 1 and 2. The horizontal axis indicates the number of sentences per template that were used in the training, and the vertical axis indicates the percentage of output units within .15 of the correct value. The networks were trained for 380,000 input sentence presentations with 0.1 learning rate and another 190,000 presentations with 0.05. This is a total of 375 – 15,000 epochs, depending on size of the training set.

training data. For example, `The hammer ate the window with the boy` would have a drastically different pattern. Given the experience the network has about hammers, eating, windows, and boys, this sentence would be very unlikely to occur, and the network would have difficulty processing it. A sentence like this could not be generated by the sentence templates of Appendix 13. It does not belong to the input space the system is trying to learn: It makes no sense in the microworld.

As a result, the FGREP system processes both the familiar and unfamiliar test sentences *at the same level of performance* (Tables 1 and 2). This is in contrast to systems using representations with precoded, fixed semantic features, such as McClelland and Kawamoto's (1986). Even though two words are functionally equivalent in the input data, in McClelland and Kawamoto's system the feature representations remain different, and the same level of generalization is much harder to achieve. On the other hand, FGREP loses the ability to distinguish between equivalent words (this problem is discussed in Section 6).

An interesting question is, how well can FGREP learn the representations and how well does it generalize if it is trained with only a small subset of the input data? This was tested in a series of simulations. Equal numbers of sentences from each template were selected randomly for the training, including all sentences in the familiar set and excluding all sentences in the unfamiliar set. When a template produced fewer sentences than needed, multiple copies were used.

The performance as a function of the training-set size is plotted in Figure 9. With very small training sets, the system learns the idiosyncrasies of the training sentences and generalizes poorly. Generalization improves very quickly as more training data is included. A critical mass is reached at around eight sentences per template, which covers approximately 10% of the input space. At this point, the system gets 96% of the unfamiliar sentence output within .15;, adding more training data does not significantly improve performance.

Even with the very incomplete training data, the final representations reflect the similarities of the words very well. Only the fine tuning of the equivalent words is lost (i.e., their representations are very close but not exactly the same). This is because these words no longer have exactly the same distribution in the training data. The conclusion is that as long as the training data constitutes a good statistical sample of the I/O space, FGREP will develop meaningful representations. The similarities are more and more coarse the smaller the training set, because the asymmetries in the data are coded in. In this particular task and data, *a sample of 10% was large enough to develop meaningful representations, which allowed the system to generalize correctly to the last 90% of the input space.*

The results in Figure 9 are actually better than those presented in Tables 1 and 2, because the training data was selected differently. In the generalization experiment, the approach was to train the system *with all templates equally.* In the baseline simulation (Section 5.1), where the training set was almost complete, the system was effectively trained with the *actual sentence distribution.* Because some templates produce 3 different sentences whereas others generate 728, different performance figures were obtained for the two cases. A more appropriate test for the baseline system is to test it on the complete set of sentences: the average error is .023, while 97% of the output units are within .15. This is the type of test that will be used in later sections.

6 ID+content: cloning synonymous word instances

6.1 Meaning and identity

The FGREP approach is based on the philosophy that *the meaning of a word is manifest in how it is used.* Learning a language is learning the use of the language elements: *Language is a skill.* An FGREP representation is defined by all the contexts where the word has been encountered, and it determines how the word behaves in different contexts. The representation evolves continuously as more experience about the word is gained. In other words, FGREP extracts the meaning of the word from its use, and encodes it into the representation.

No two words are used exactly in similar ways in real-world situations and, in principle, the meanings of two words are always distinguishable by their use. If there is enough difference in the usage, the FGREP process will develop different representations for the words. However, an artificial intelligence system can be exposed only to limited experience, and keeping the representations separate becomes a problem. It is unwieldy to try to generate training sets that would allow enough differences to develop between similar word representations. For example, when the FGREP system reads `The boy hit the girl with the ball` (Figure 2), it produces an output pattern that is very close to correct, but it is impossible to tell just by looking at the patterns at each output assembly, whether the humans in the actor and patient slots are `man, woman, boy`, or `girl`, and whether the instrument/modifier is `ball, hatchet`, or `hammer`. Their representations are almost exactly the same.

Word discrimination of the system was measured by finding the closest lexicon representation (in Euclidian distance) for each output assembly and counting how often this was the correct one. Even though the system has learned the mapping task very well (97% of the output units are within .15), it produces a pattern closest to the correct output word only about 77% of the time. A closer look reveals that the output words that have unique meanings are correct, but only 59% of the words that have synonyms (i.e., equivalent words such as `man, woman, boy, girl`) are closest to the correct word. The best word discrimination was achieved at around the 100th epoch during training, when the network had learned the case-role assignment task fairly well but the representations had not yet completely converged, providing enough differences to keep the representations separate. Even in the end, the discrimination between synonymous words remains above chance level, because the training data were not completely symmetric, and minute differences remain in the representations. Also, the generalization networks of Figure 9, which were trained with even more incomplete data, can separate the words much better. We arrive at the curious conclusion that the more extensively the system has experienced the use of the input items, the worse it can keep track of the identities of items that have similar meanings.

The identity problem is even more fundamental in cases where it is necessary to create several distinct instances of the same item temporarily. For example, in `The man who helped the boy blamed the man who hit the boy`, it is crucial to know that the man-who-helped is the same as the man-who-blamed, but different from the man-who-hit. Yet there is no way of telling this from the meaning of `man`. It seems necessary to complement the meaning of the item, as extracted from its use, with a surface-level tag that provides a distinct identity for the item.

6.2 Composing instances from ID and Content

The simplest way to maintain distinct identities for each word is to designate a subset of representation components for this purpose. The representation vector now consists of two parts: the content part, which is developed via the FGREP process and which encodes the meaning of the word, and the ID part, which is unique for each instance of the same meaning.

Before training, a set of *prototype words*, that is, words that we later want to make copies of, is selected (e.g., `HUMAN, FOOD, UTENSIL`). During training, the units within the ID part of the prototype word representation are set up randomly for each input presentation, and the network is required to produce the same ID pattern at its output. In effect, the network is trained to process any ID pattern in a prototype word by passing the ID parts unchanged from input to output assemblies.

After training, a number of separate *instances* of the prototype words are created by concatenating a unique ID with the content part of the developed prototype. These new words (e.g., `John, Mary, Bill` for the prototype word `HUMAN`) are then added to the lexicon, and they can be used for input and output. This *ID+content* technique is important for four reasons:

1. It makes it possible to deal with a large and open-ended set of semantically equivalent names without confusing them. In other words, we can create several tokens from the same basic type, and the system will know how to process them. This seems to be a prerequisite for modeling symbolic thought and logical reasoning (Section 12.3).
2. Even if the system has been trained with only a small number of distinct meanings, it can process (e.g., parse) a much larger vocabulary *approximately*. The representations for the new words are cloned from the existing meanings (Section 6.4).
3. New meanings can be incrementally learned by using the cloned representation as the initial guess and letting it acquire finer semantic content in subsequent training (Section 11.3).
4. It allows us to model creation of temporary and fictitious characters in story understanding. For instance, `John` in the beginning of the story is simply an instance of human (yet separate from other humans), until we learn more about him.

The ID part does not need to have any intrinsic meaning in the system. It is a surface-level tag, whose function is to distinguish the word from all other similar words. However, there are two sources from which the ID pattern could arise:

1. It could be based on the acoustical or orthographical qualities of the word itself. For example, even though `pot` and `pan` may have nearly identical meanings in a kitchen context, at the surface they are acoustically and orthographically distinct, and can be kept separate.
2. It could be based on the sensory properties of the word referent. For example, the man-who-hit would have a different image attached to it than the man-who-helped. For this reason, the ID+content technique can be thought of as an approximation of sensory grounding (Section 12.3).

6.3 Performance with cloned synonyms

The ID+content technique was tested by developing a prototype word representation for each of the word equivalence classes: `HUMAN` for {`man`, `woman`, `boy`, `girl`}, `UTENSIL` for {`fork`, `spoon`}, `PREDATOR` for {`wolf`, `lion`}, `GLASS` for {`plate`, `window`}, `GEAR` for {`ball`, `hatchet`, `hammer`}, `BLOCK` for {`paperwt`, `rock`} and `FOOD` for {`cheese`, `pasta`, `carrot`}. The words in braces were replaced by their prototype in the training data, and duplicate sentences were removed. The new training set consists of 207 different sentences. The first two units of the representation were designated for the IDs and the content part consisted of the 10 remaining units. The network was trained with 0.1 learning rate for 2,000 epochs and with 0.05 until epoch 5,000, which took approximately the same time as training without cloning.

After training, each prototype was cloned to cover the original vocabulary, choosing binary values for the two ID units. The resulting representations reflect the categorization as before, and have the same processing characteristics. However, representations within each category are now more distinct.

Using the cloned representations, the system is able to keep the equivalent words separate. Ninety-seven percent of all output words are now correct, with 98% of the words with synonyms. The error statistics are slightly worse (96% within .15, with .038 average error) partly because the ID components of the instances consist of extreme values 0 and 1, which are harder to achieve than midrange values.

With this particular task and data, the representations in neighboring equivalence classes, such as `GEAR` and `BLOCK`, become very similar. If the ID patterns are very dissimilar, items are sometimes confused across categories with items having similar ID patterns. In other words, there is a danger that the ID is used as a basis for generalization. This can be avoided by keeping the ID part small, and the ID patterns different for different prototypes (e.g. random).

6.4 Extending the vocabulary

The ID+content technique can be applied to any word in the training data, and in principle, the number of clones per word is unlimited. This technique allows us to significantly increase the size of the vocabulary while having only a small number of *semantically* different words at our disposal. Even though in principle each word has a unique meaning, this allows us to *approximate the meanings of a large number of words by dividing them into equivalence classes.*

This capability is important from the practical point of view when trying to build an AI system to process natural language in the real world. Training the system with a large vocabulary is expensive. All the fine differences in usage of, for example, `give` and `donate`, need to be included in the training data. The ID+content technique allows us to train the system with meanings that are most crucial for the task, and build a larger vocabulary (that the system needs in order to deal with the real world) by expanding on those few basic meanings.

For example, the system can process sentences like `John downed the lasagna with a plastic-fork` and `Mary ate the chicken-soup with a silver-spoon`. The fine differences in meaning between these two sentences are available only to the external observer. The system would not understand the different connotations of these words, because it was not trained with them. Internally, the system processes only one event: `Human ate the food with a utensil`. Although this is not exactly right, it captures the essential content of the sentences. This approach allows the system to perform its task (e.g., answer questions) robustly, even if it does not capture all the subtleties of the words.

A similar approach is, in fact, taken by NLP systems based on conceptual dependency theory (Schank and Abelson, 1977). For example, both `run` and `walk` map to a hand-coded structure

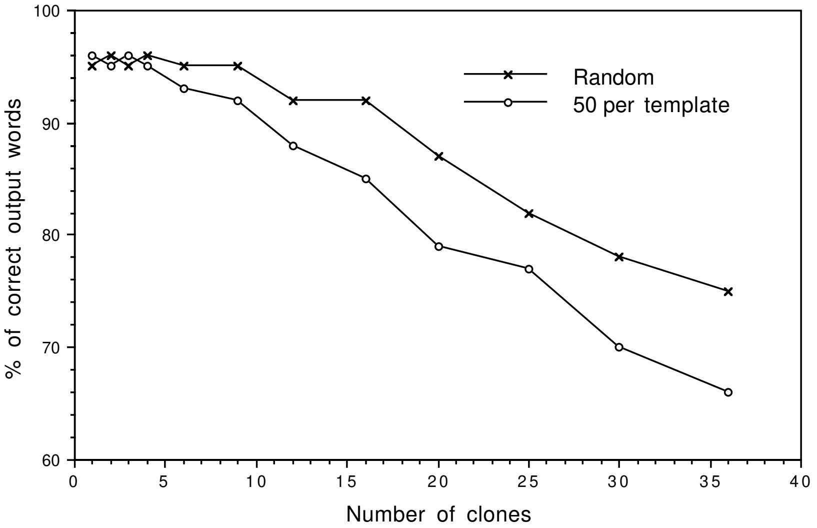

Figure 10: **Word discrimination of basic FGREP with extended vocabulary (ID+content technique).** The horizontal axis shows the number of clones created for each word, and the vertical axis indicates the percentage of correct output words. The random data set consisted of 1,000 sentences selected randomly among the set of all sentences, whereas the 50-per-template data set consisted of an equal number of randomly chosen sentences from each template. A 0.1 learning rate was used for the first 3,000 training epochs and 0.05 for another 3,000. The error average is not affected by the number of clones. The average error for the random data set was .035, with 97% of the output within .15, and for the 50-per-template data set .040 and 95%.

involving the primitive act PTRANS, which stands for physical transfer. This approach allows a symbolic NLP system to handle many inputs, at the cost of losing the subtle connotations of similar words.

However, the FGREP with ID+content approach also allows the connotations for similar words to be learned. A small initial vocabulary allows us to *bootstrap* the system with artificial training data. We can then extend the vocabulary with new instances of the basic meanings, and let them acquire more accurate semantics through FGREP adaptation in the actual task (Section 11.3).

Figure 10 plots performance with an increasing number of clones for each word. The system was trained with the same data as before, but this time all words (except the blank) were cloned. In this experiment the ID values were spaced out evenly in the unit square. Uniformly distributed random values produced similar, only slightly weaker results. The space around each ID decreases inversely proportional to the number of clones. In other words, as the number of clones increases, the IDs become more and more similar, making the discrimination harder.

As can be seen from Figure 10, discrimination degrades approximately linearly as a function of clones. With 16 different clones represented in two-unit IDs, the system still produces correct words 93% of the time. This is remarkable because *the number of different sentences grows polynomially, proportional to the fourth power of the number of clones.* Without cloning, there are 210 different sentences; 4 clones produce 27,024, 16 clones produce 5,495,040 and 36 clones produce 134,401,680 different sentences.

Cloning word instances is very much like generating new symbols with the LISP function `GENSYM`. In addition, *the instances automatically have intrinsic meaning coded in them.* The processing knowledge is separate from the symbols that can be processed. With linear cost, the system can process a combinatorial number of inputs (Brousse and Smolensky, 1989) in a nontrivial task.

7 Processing sequential input and output: The recurrent FGREP module

The sentence-processing architecture presented in the preceding sections relies on highly preprocessed input. An external supervisor determines the syntactic constituents of each sentence, which is a nontrivial task in itself, and the sentence is represented in terms of these constituents in a fixed assembly-based representation.

In this section we show how these requirements can be relaxed. A recurrent extension of the FGREP architecture is presented, allowing us to deal efficiently with sequential input data. The architecture is used to form case-role representations directly from word-by-word sequential natural language without the need for preprocessing.

7.1 Representing constituent structures

When data is represented with *role-specific assemblies*, as in Hinton (1981), McClelland and Kawamoto, (1986), and in the FGREP architecture presented so far, the constituents of a complex data item can be correctly interpreted only in their appropriate assemblies. An assembly must be reserved for each possible constituent, whether it is actually present in the input or not. For example, there is an "object" section and a "with" section in each sentence representation, even though not all sentences have these elements. Representation of structure becomes a serious problem when we want to scale up and represent, for example, stories. The number of different constituents is enormous, although only a few of them are present at any one time. Separate assemblies have to be reserved for each of them. This leads to a combinatorial explosion in the size of the representations.

On the other hand, role-specific assemblies preserve the high-dimensional relations of the constituents. The relation of a constituent to all other constituents is well defined as soon as it is placed in a specific assembly. All constituents of the representation are available simultaneously and in parallel, which makes efficient higher order inferencing possible. This powerful representation style is suitable and desirable for internal representation, which is usually not severely limited by bandwidth.

Assembly-based representation can sometimes be made more efficient by letting part of the representation determine how the rest of the assemblies should be interpreted. The representation consists of two parts: The *base* part determines the semantics of the *body* of the representation. This use of *data-specific assemblies* makes the most sense when the data consists of disjoint classes with distinct constituent roles, such as stories based on different scripts (see Section 10.2). The technique reduces the number of assemblies needed, while still providing the same representational power.

Input and output channels, however, typically have very limited bandwidth. In certain cases, some of the assembly-based representational power can be given up to achieve more efficient communication. If the relations are sufficiently simple, information about partial ordering of the elements may be enough.

If there is a plausible way to "linearize" the data structure, representing it as a sequence is an efficient solution. There is no need to represent missing constituents, and structural information is conveyed by the order of constituents. In other words, input and output of complex data takes place exactly as in natural language, where complex thoughts are transformed into a sequence of words for transmission through the narrow communication channel. Consequently, natural language input and output is most naturally and efficiently represented in this manner.

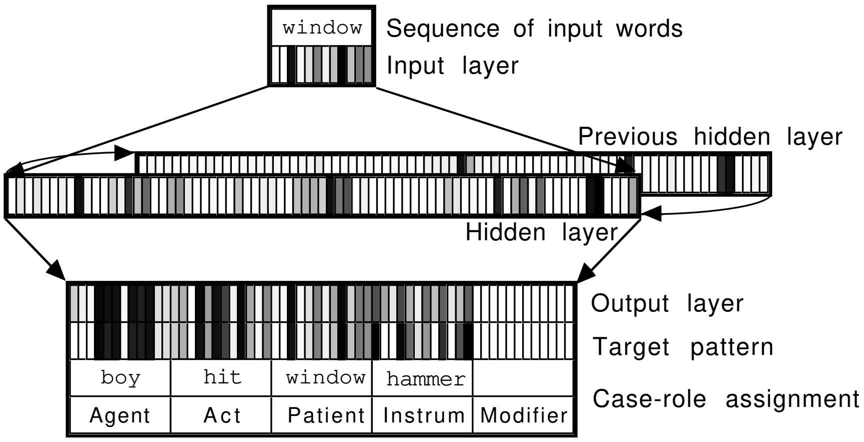

Figure 11: **Snapshot of recurrent FGREP simulation.** The network is in the middle of reading `The boy hit the window with the hammer`. The hidden layer pattern is saved after each step in the sequence, and used as input to the hidden layer during the next step, together with the actual input.

7.2 Recurrent FGREP module

Recurrent FGREP, the extension of FGREP to sequential input and output (Miikkulainen and Dyer, 1989b), is based on Elman's Simple Recurrent Network (Elman, 1989, 1990; see, also, Jordan, 1986; Servan-Schreiber et al., 1989; St. John and McClelland, 1990). 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 11). 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 both for reading a sequence of input items into a stationary output representation, and for generating an output sequence from a stationary input. In a *sequential input network*, the input pattern changes at each time step, while the target pattern stays the same. This network forms a stationary representation of the input sequence. In a *sequential output network*, the input is stationary, but the target pattern changes at each step. The network produces a sequential interpretation of its input. The error is backpropagated and weights are changed at each step. Both kinds of recurrent FGREP networks also develop representations in their input layers.

7.3 Case-role assignment from sequential input

Recurrent FGREP was tested with the same case-role assignment task and data as before, with cloning of synonymous words. The network architecture is illustrated in Figure 11. More resources are required to assign case roles from sequential input than in the preprocessed case. The hidden layer now also serves as a sequence memory, and more capacity is needed. Training times are longer because the network now has to learn sequences, and backpropagation takes place after each item in the sequence. Our network had 75 units in the hidden layer, and it was trained with 0.1 learning rate during the first 1,500 epochs, 0.05 until 2,000 and 0.025 for another 100 epochs.

The input consists of the actual words in the sentence templates (Table 5 in Appendix 13), now also including the words `the` and `with`. The main purpose was to show that the network can process raw natural language, but the network also learns to disambiguate expectations based on these words (Section 7.4). The complete case-role representation is used as the training target throughout the sequence.

Word by word, the representations are fetched from the lexicon and loaded into the single input assembly. The activity is propagated to the output layer. 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. As more words are read in, the complete case-role representation of the sentence gradually emerges at the output. Each sentence is ended with a period, *which forms its own representation in the lexicon.* The case-role representation is complete after the period is input.

Processing word sequences does not significantly degrade the performance: the system now outputs 97% of the words correctly, with 95% of the words with synonyms. Ninety-three percent of the output values are within .15, and average error is .045. The representations look very much like in the stationary case, with the same processing properties.

7.4 Sequential expectations

When the input is sequential, expectations embedded in the word representations are clearly demonstrated. Expectations arise in the assemblies for unspecified case roles (Figure 11). During training, the network is required to produce the complete output pattern after each step in the sequence, even before it is possible to recognize the sequence unambiguously. As a result, the output patterns at each step are averages of all possible event sequences at that point, weighted by how often they have occurred. After the next word is input, some of the ambiguities are resolved and correct patterns are formed in the corresponding assemblies. Often the sentence representation is almost complete before the sentence has been fully input.

In Figure 11, the network has read `The boy hit the window`, and has unambiguously assigned these words to the agent, act, and patient roles. The instrument and modifier slots indicate expectations. At this point it is already clear that the modifier slot is going to be blank, because only human patients have modifiers in our data (Table 5). Most probably, an instrument is going to be mentioned later, and an expectation for a hitter (an average of all possible hitters) is displayed in the instrument slot. If `with` is read next, a hitter is certain to follow, making the pattern even stronger. Reading a period next would instead clear the expectation in the instrument slot, indicating that the sentence is complete without this constituent.

The expectations emerge automatically and cumulatively from the input word representations. This feature can be used, among other things, to fill in missing input information (Section 10.7), and to guess meanings of unfamiliar words (Section 11.3).

At this point it is appropriate to point out the differences of the FGREP sentence-processing architecture with that of St. John and McClelland (1990). Their research has concentrated on combining syntactic, semantic, and thematic constraints in sentence comprehension, and how this knowledge can be coded into the network. The network reads a sequence of phrases into a sentence gestalt, which is basically a hidden layer representation of the whole sentence. The gestalt can then be queried for roles. For example, the sentence gestalt for `The busdriver ate the food`, formed by the first part of the system, is input to the second part of the system together with the representation of a query, for example, `patient?`. The final output is the representation of the answer, patient=`steak`. Because bus drivers usually eat steaks in the data, the system produces this as the answer.

St. John and McClelland's model shows how useful thematic knowledge of this kind can be injected into the system, by training it with queries and by handcoding the microfeature representations. The model can answer queries that require information beyond what is strictly provided by the sentence. However, the system cannot be easily used as a modular building block in more complex processing. The network's knowledge is *implicit* and *noncompositional*, and can be brought out only by querying the gestalt in the system. To support modularity, the knowledge structures would need to be explicit in the output representation, as in the recurrent FGREP module.

8 Limitations of FGREP

The FGREP network codes the meaning of an input item into its representation, and the meaning can be demonstrated by creating expectations, as was shown in Sections 5.5 and 7.4. However, the representations can be fully interpreted only by the network that developed them. For the rest of the world, the representations for `sheep` and `wolf` look similar, but there is no way to tell that they are both animals. Just by looking at a representation it is impossible to extract the features coded in it.

However, the representations do reflect the similarities between concepts, and thus form a fairly good basis for other tasks. It is possible to train other systems to use the representations fairly efficiently, although they are optimal only in the task they were developed in. Semantics of individual representations are not portable, and must be redeveloped in the new task.

The FGREP mechanism is also somewhat limited by the fixed assembly-based internal representation. It is possible to use sequential input or output, but the internal representation must be laid out on a fixed number of assemblies. The assemblies can be data-specific, which significantly increases their representational power. However, multiple fillers are still a problem. Sentences like `John, Mary, Bill, and Susan went to the beach` cannot be represented on a fixed number of slots without posing a hard limit on the number of possible agents.

In the previous sections, the FGREP mechanism was demonstrated in parsing simple sentences. These sentences could be easily represented in case-role assignment form and processed by a single recurrent FGREP module. More complex parsing with a single module is also possible, but it is better to build a parser from several modules. For example, two hierarchically organized modules can read sentences with multiple hierarchical relative clauses into a canonical internal representation (Miikkulainen, 1990c). Building from FGREP modules is a powerful technique, and it is discussed in detail in the following sections.

9 A composite task: Paraphrasing script-based stories

The recurrent FGREP modules were designed to be used as building blocks in more complex cognitive systems. An obvious task is the scale-up of sentence processing to the next higher level: processing stories. The goal here is to train a neural network system to produce full paraphrases of incompletely specified script-based input stories. An example story:

> `John went to MaMaison. John asked the waiter for lobster. John left a big tip.`

This story mentions only three events about John's visit to MaMaison. A human reader can fill in a number of events, which certainly (or most likely) must have occurred, and he can paraphrase the story in more detail. In doing this, he uses his general experience about events that occur during a visit to a restaurant:

> `John went to MaMaison. The waiter seated John. John asked the waiter for lobster. John ate a good lobster. John paid the waiter. John left a big tip. John left MaMaison.`

Schank and Abelson (1977) suggested that this kind of knowledge about everyday routines could be organized in the form of *scripts*. Scripts are schemas of often-encountered, stereotypical

RESTAURANT SCRIPT		
FANCY-RESTAURANT TRACK		
Causal Chain:	**Roles:**	
Entering	Customer	= `John`
Seating	Restaurant	= `MaMaison`
Ordering	Food	= `lobster`
Eating	Taste	= `good`
Paying	Tip	= `big`
Tipping		
Leaving		

Table 3: **Representation of a script-based story as a causal chain and role bindings.**

sequences of events. Common knowledge of this kind makes it possible to efficiently perform social tasks such as visiting a restaurant, visiting a doctor, shopping at a supermarket, traveling by airplane, and attending a meeting. People have hundreds, perhaps thousands of scripts at their disposal. Each script further divides into different variations, or tracks. For example, there is a fancy-restaurant track, fast-food track, and a coffee-shop track for the restaurant script, each with slightly different events.

In symbolic natural language processing a script is represented as a causal chain of events with a number of open roles (Cullingford, 1978; DeJong, 1979; Dyer et al., 1987; Schank and Abelson, 1977). A script-based story is an instantiation of the script with specific role bindings. Applying knowledge about scripts to a story requires identifying the relevant script and filling in its roles with the actual constituents of the story (Table 3). Once the script has been recognized and the roles have been instantiated, the sentences are matched against the events in the script. Events that were not mentioned in the story but are part of the causal chain can be inferred, as well as certain fillers of roles that were not specified in the story. For example, it is plausible to assume that John ate the lobster and the lobster tasted good. The story can then be paraphrased in full from its representation.

The causal chain is a summary of all restaurant experiences and remains stable, whereas the role bindings are different in each application of the script. This distinction suggests a neural network approach for processing scripts. The causal chain of events is learned from exposure to a number of restaurant stories, and is stored in the long-term memory of the network (i.e., in the weights). The role bindings are processed as activity patterns in data-specific assemblies.

10 Connecting the building blocks in DISPAR

10.1 System architecture

The DISPAR system (DIStributed PARaphraser; Miikkulainen and Dyer, 1989b) consists of two parts (Figures 12 and 13). The first part reads the story, word by word, into the internal representation, and the second part generates a word-by-word fully expanded paraphrase of the story from the internal representation. Each part consists of two recurrent FGREP modules, one for reading/generating the word representations and the other for reading/generating sentence representations. We call these modules the sentence-parser/sentence-generator and the story-parser/story-generator networks. During performance, the whole system is a chain of four networks, each feeding its output to the input of the next network. The input and output of each network consists of distributed representations of words that match the ones stored in the global lexicon.

10.2 Performance phase

Let us present the system with the first sentence of the example story, `John went to MaMaison` (Figure 12). The task of the sentence-parser network is to form a stationary case-role representation of this sentence, as described previously. Words are input to this network one at a time, and the representation is formed at the output layer. After a period is input, ending the first sentence, the final activity pattern at the output layer is copied to the input of the story-parser network. The story parser has the same structure as the sentence parser, except that it receives a sequence of the case-role representations as its input, and forms a stationary slot-filler representation of the whole story at its output layer.

The final result of reading the story is the slot-filler assignment at the output of the story-parser network. This is the representation of the story in terms of its role bindings, with two additional assemblies specifying the script and the track. The role assemblies stand for different roles depending on the script. In our example, the representation consists of script=`restaurant`, track=`fancy`, customer=`John`, restaurant=`MaMaison`, food=`lobster`, taste=`good`, and tip=`big`. This technique makes the best use of the fixed-size assembly-based representation. The assemblies are data-specific, rather than role-specific, and their interpretation varies with the data. The script assembly constitutes the base of the representation. For example, if script=`$travel`, then the fourth assembly (R/Food in Figures 12 and 13) will serve the role of T/Origin, the origin of the trip.

The slot-filler representation completely specifies the events of the script. The second part of the system (Figure 13) is trained to paraphrase the story from this internal representation. The idea is simply to reverse the reading process.

The story-generator network (Figure 13) receives the complete slot-filler representation as its input, and generates the case-role assignment of the first sentence of the story as its output. This pattern is fed to the sentence-generator network, which outputs the distributed representation of the first word of the first sentence. The hidden layer of the sentence-generator network is copied to the previous-hidden-layer assembly and the next word is output.

After the last network produces a period, indicating that it has completed the sentence, the hidden layer of the story-generator network is copied into its previous-hidden-layer assembly, and the story-generator network generates the case-role representation of the second sentence. The process is repeated until the whole story has been output.

10.3 Training phase

The modular architecture provides a major advantage in training these networks. The tasks of the four networks are separable, and they can be trained separately (e.g., on different machines) as long as compatible I/O material (i.e., the same stories and the same lexicon) is used. The networks must be trained simultaneously, so that they are always using and developing the same representations.

The lexicon ties the separated tasks together. Each network modifies the representations to improve its performance in its own task. The tasks have conflicting requirements, and the representations 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 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 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.

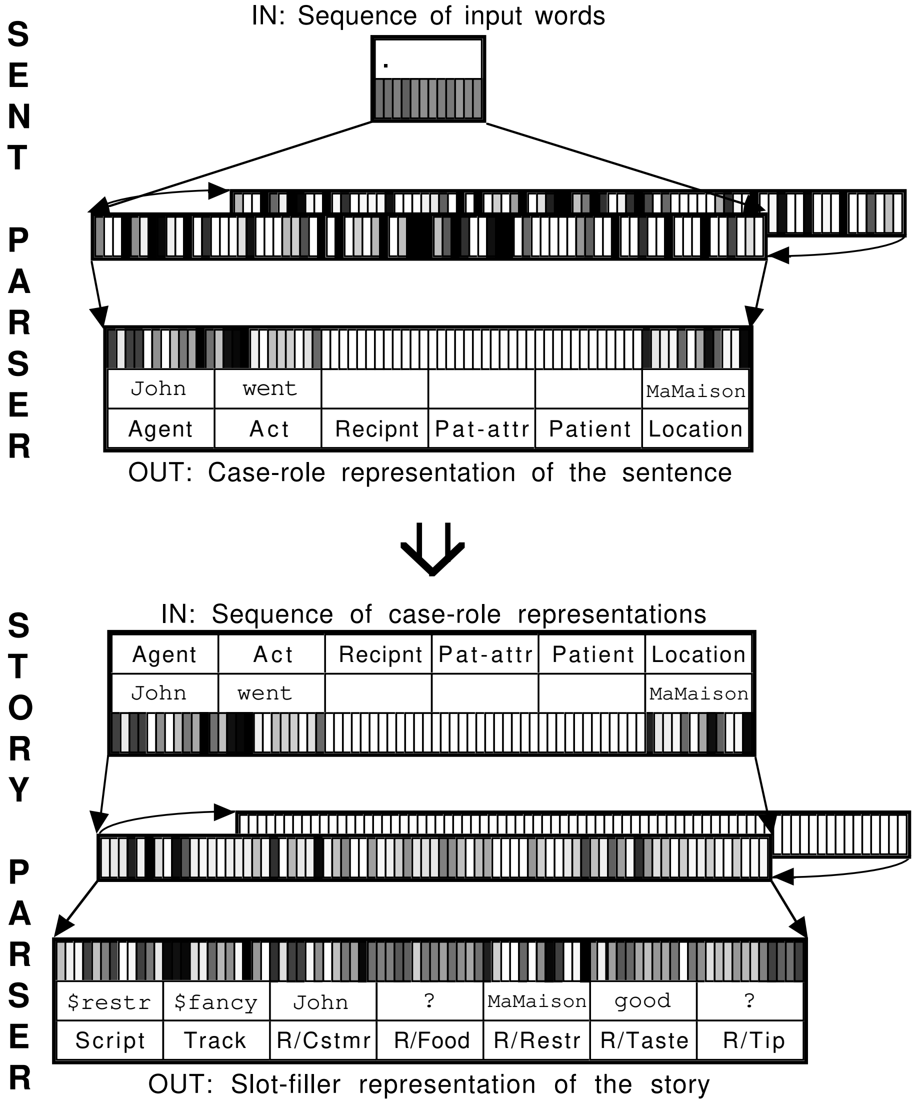

Figure 12: **Networks parsing the story.** A snapshot of the simulation after the first sentence of the example story has been read. A period, ending the first sentence, is in the input assembly of the sentence parser. The script and the track have already been identified and a number of expectations about the role bindings are active at the output of the story parser. The role names (R/...) are specific for the restaurant script.

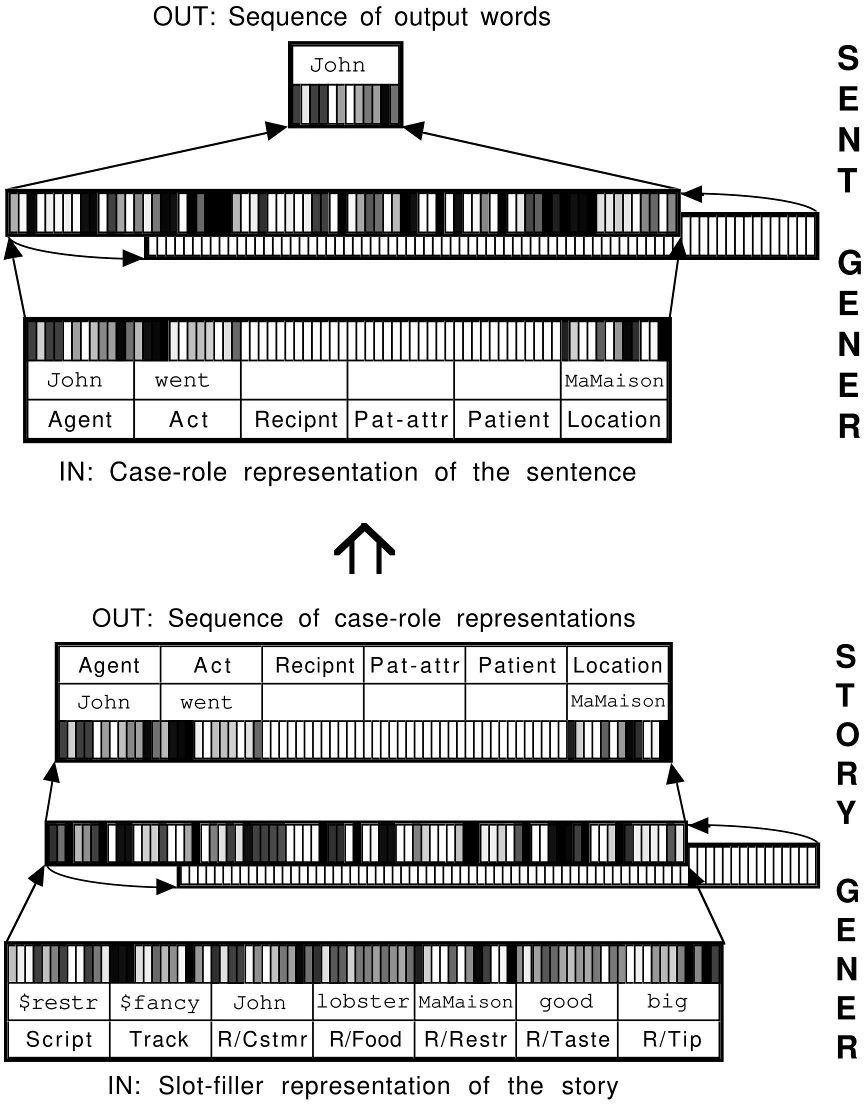

Figure 13: **Networks generating the paraphrase.** A snapshot of the simulation, where the story generator has produced the case-role representation of the first sentence, and the sentence generator has output the first word of that sentence. The previous hidden layers are blank during the first output.

10.4 Story data

The DISPAR system was trained to paraphrase stories based on restaurant, shopping, and travel scripts. There were three tracks to each script, with four to five open roles. The story templates for each track are listed in Appendix 13. Actual stories are formed from these templates by specifying the ID part for each prototype word that appears in the template.

The need for the script and track assemblies in the story representation is evident from the data. The script is used to specify how the patterns in the role assemblies should be interpreted (i.e., what the roles are). The track is necessary because the order of events is slightly different in different tracks. Without the track slot it would be necessary to represent the differences as additional role bindings, which would be inefficient and unnatural. The script and track patterns are treated just like words in the system. Their representations are stored in the lexicon and modified by FGREP during training.

The restaurant tracks contain some interesting regularities: the food is always good in a fancy-restaurant, and always bad in a fast-food restaurant. In coffee-shop restaurant, the size of the tip correlates with the food quality: good food ⇔ big tip, bad food ⇔ small tip. These regularities were set up intentionally; the system should be able to use them to fill in unspecified roles.

10.5 Simulations

The four networks were trained separately and simultaneously with compatible I/O data. Each story was processed as if the networks were connected in a chain, but the output patterns, which are more or less incorrect during training, were not directly fed to the next network. Instead, they were replaced by the correct patterns, obtained by concatenating the current word representations taken from the lexicon. This way, each network was trained over the same number of epochs with compatible data. For an alternative training method, where the networks are trained on separate machines, see Miikkulainen and Dyer, 1989b).

The learning rate was gradually reduced from 0.1 to 0.0025 over a total of 20,500 epochs (0.1, 0.05, and 0.025 for 5,000 epochs each, then 0.01 for 2,500 epochs, and 0.005 and 0.0025 for 1,500 epochs each). Word representations consisted of 12 units and each network's hidden layer of 75 units. Two units were used for the IDs.

In the training, the IDs for the prototype words were assigned randomly for each story. In the performance phase, two instances were created from each prototype word, using orthogonal values (1 0) and (0 1). The set of test stories was put together from all combinations of the instances, 96 stories altogether (including the two different versions of tip for $fancy-restaurant, taste/tip for $coffee-shop-restaurant, and distance for $train-travel). Performance of the system was tested with two sets: (1) complete stories, which included all sentences in the template, and (2) incomplete stories, containing only three sentences.

10.6 Representations

There are very few similarities in the resulting representations. The vocabulary in the example stories is fairly large compared to the number of different stories (86 word prototypes vs. 9 story templates), and most words have very distinct usage. As a result, the word representations also become distinct. Only the words for different types of restaurants, foods, shops, shopping items, and travel destinations are faintly similar. The system has no trouble keeping the words separate with this data.

The system learns to output the period quite early in the training. Very seldom is a sentence produced without a period at the end. On the other hand, in the early stages of training, sentences

Network	All words	Instances	$E_i < .15$	E_{avg}
Sentence parser	99/99	99/99	99/98	.017/.019
Story parser	99/96	99/97	96/93	.023/.041
Story generator	99/99	99/97	97/97	.021/.027
Sentence generator	99/99	99/97	96/95	.037/.044

Table 4: **Performance of DISPAR networks with complete/incomplete input stories.**

are often ended prematurely with a period, and after that, only periods are output. This happens because the representation for the period is modified at the end of every input sentence, where reading it should have very little effect on the output. Its representation becomes the most neutral representation, that is, it contains many components close to 0.5 (see, e.g., Figure 12). This representation makes the period a default output when the network does not know what word to generate next.

The period is treated in the system just like a word, with a semantic representation of its own, formed automatically from the input examples. It seems that this approach could be used to deal with the semantics of punctuation in general. The network develops a representation for each punctuation symbol according to how it is used. The representation encodes information about possible contexts, and affects how the rest of the input is interpreted.

10.7 Paraphrasing

The training corpus consisted of complete stories, and the system was trained to reproduce a story exactly as it was input. During performance, the networks are connected in a chain, the output of one network feeding the input of the next. Table 4 presents performance numbers for each network in this task (numbers before /). Interestingly, the errors do not accumulate very much in the chain. Because the representations are distributed and redundant, noise in the input is efficiently filtered out, and each network performs approximately at the same level. In the output story, 99% of the words are correct, including 99% of the cloned words. In other words, the system produces the right customer, food, store, destination, and so forth, 99% of the time.

However, error clean-up does not take place in the ID units because they are nonredundant. The rest of the representation gives no clue about what the values of the IDs should be, and an error made in an ID unit early in the chain cannot be corrected later. As a result, the system is more likely to confuse instances of the same type rather than words with different meaning. This side effect is interesting because it seems to match human behavior. Role bindings based on syntactic constraints are harder than ones supported by a strong semantic component.

Given that DISPAR was trained to reproduce its input story, an interesting question is how well it can fill in the events when the input story consists of only a few sentences. The system performs very well in this respect (Table 4, numbers after /). There is very little degradation in performance compared to the complete stories: 99% of output words are correct, including 97% of the cloned words. The story parser introduces some error to the flow of information, but much of it is filtered out by the story generator. Incomplete stories that uniquely specify a story are paraphrased correctly to their full extent. The quality of the food (`good` or `bad`) is inferred if the size of the tip is mentioned in the story, and vice versa. If there is not enough information to infer the filler for some role, an activity pattern is produced which is intermediate (a weighted average) among the possible choices. This follows directly from the tendency to generate expectations.

For example, if the story consists only of the sentence `John went to MaMaison`, food quality can be inferred (`good`, because `MaMaison` is a fancy restaurant), but an intermediate representation develops in the food slot (Figure 12). In paraphrasing the story, it seems as if one of the possible fancy foods is chosen at random. Especially interesting is, that once a role binding (e.g., food=`steak`) is selected in an earlier event, even if it is incorrect, it is usually maintained throughout the paraphrase generation. The choice is consistent throughout the story, because all sentences are generated from the same pattern in the food slot. Thus, DISPAR *performs plausible role bindings:* an essential task in high-level inferencing and postulated as very difficult for PDP systems to achieve (Dyer, 1991).

In general, it seems that a network which builds a stationary representation of a sequence might be quite sensitive to omissions and changes. Each input item is interpreted against the current position in the sequence by combining it with the previous hidden layer. If items are missing, it seems that the system could very easily lose track. But this is not the case in the story parser. The network was trained always to shoot for the complete output representation, which means that most of the work is done when the first sentence is read in. Subsequent sentences have very little effect on the hidden layer pattern, and consequently, *omissions are not critical.*

Filling in the missing items is a form of generalization. A similar generalization in a non-sequential network (e.g., such as described in section 5) would be required when a number of input assemblies are fixed at 0.5, the “don’t care” value. The strong filling-in capability of DISPAR is due to the fact that there is very little variation in the training sequences. But it is exactly this lack of variety that makes up scripts: They are stereotypical sequences of events. Interestingly, it follows that *filling in the unmentioned events is an easy task for a sequential network system* such as DISPAR.

Reading in the first sentence is analogous to script instantiation in symbolic systems. If the first sentence is missing, processing the sequence becomes much harder. The network does not know how to combine an intermediate sentence with a blank previous hidden layer. The output improves as more sentences are read in, although the finer details usually remain inaccurate.

Once the script has been instantiated and the role bindings fixed, there is no way of knowing which events were actually mentioned in the story. What details are produced in the paraphrase depends on the training of the output networks. This result is consistent with psychological data on how people remember stories of familiar event sequences (Bower et al., 1979). The distinction of what was actually mentioned and what was inferred becomes blurred. Questions or references to events that were not mentioned are often answered as if they were part of the original story.

10.8 Further extensions to script-processing architecture

Question answering can be implemented as a separate module that receives as its input the slot-filler representation of the story, together with the representation of the question that has been parsed sequentially by the sentence-parser network (Miikkulainen, 1990a). The module generates a case-role representation of the answer sentence, which is then output word by word by the sentence-generator network.

A more complete script-processing system also has an episodic memory for the stories. After the story has been parsed, the slot-filler representation is saved in the episodic memory. Later, it can be retrieved with a partial story representation, such as a question referring to that story, as a cue. The organization for the memory can be formed in a self-organizing process, and the result reflects the taxonomy of script-based stories (Miikkulainen, 1990a).

Pronoun reference is not a particularly hard problem in understanding script-based stories. People do not get confused when reading, for example, `The waiter seated John. He asked him for lobster. He ate it.` The events in the story are stereotypical, and once the role binding has been done, the reference of the pronoun is unambiguous. It should be possible to train the network to deal with pronouns in the stories. Some of the occurrences of the referents can be replaced by `he`, `she`, or `it`, and very likely the representations for these words will develop into a general actor and object.

It might be possible to develop a mechanism for representing multiple scripts and their interactions (e.g. a telephone script or robbery script occurring within a restaurant script). One approach would be to use distributed representations for the roles and the scripts, instead of fixed, designated assemblies (Dolan, 1989; Dolan and Smolensky, 1989). Instantiation of a script would now have the form of a tensor product "cube." One face of the cube would stand for the script and track, one for the role, and one for the filler. It should be possible to represent multiple simultaneously active scripts in the same cube. Scripts could be partially activated and their boundaries would be less rigid.

In training DISPAR, an external supervisor has to decide on a fixed set of slots (such as R/Customer, T/Origin, etc.), and tell the system what the correct fillers should be for each story. This way, the scripts are implicitly coded into the training data. However, hierarchical feature maps can build a script taxonomy even when example stories are represented directly by a concatenation of the sentence representations (Miikkulainen, 1990d). Perhaps the taxonomic knowledge could be extracted from the hierarchical feature maps and used to form internal representations for the stories. This way, the training data could be formed automatically, without preconceived notions about possible scripts.

11 Towards more advanced models

11.1 Making use of modularity

Most of the PDP models have relied on capabilities of homogeneous architectures, such as basic backpropagation networks, which compute a simple mapping from input to output. Complexity of the mapping can sometimes be reduced by splitting the input space into regions, and assigning different subnetworks to different regions (Jacobs, 1990; Waibel, 1989). Alternatively, modules can be developed that process different subsections of the same input representation (Ritter, 1989).

Modularity in high-level cognitive processing has a somewhat different character. A complex cognitive task can often be broken down into simpler subtasks, which are performed sequentially, one task depending on the output of another (Ballard, 1987; Minsky, 1985). In many cases this will result in stronger models even when the task could be performed with homogenous networks.

For example, Harris and Elman (1989) demonstrated how a simple recurrent network can learn to represent co-occurrence relationships (i.e., role bindings) in script-based stories. The stories (various instantiations of scripts) are input word by word, and the network is trained to predict the next word. The fixed words in the script are predicted very well, but the correct role fillers are predicted only if they were recently mentioned in the story. However, this problem does not arise in DISPAR. The hierarchical organization of parser modules reduces the amount of information that each subnetwork has to represent in its hidden layer, and consequently, remembering the correct fillers is much easier.

In general, building the system from *hierarchically organized separable modules* provides two major advantages: (1) the task is effectively divided into subgoals, which is an efficient way to reduce complexity (Korf, 1987; Minsky, 1963), and (2) the system is forced to develop meaningful internal representations, which makes it possible to add more functions to the system in a modular way.

A simple comparison of a hierarchical network and a "flat" network was devised to demonstrate the first point. The hierarchical system consisted of the parser networks of DISPAR, with sentence case-role assignment as an intermediate representation. The flat network consisted of a single recurrent FGREP module with 126 hidden units, giving approximately the same number of connection weights. Both systems were trained to read script-based stories word by word into the slot-filler representation. The hierarchical system achieved a satisfactory level of performance about twice as fast as the simple system. The simple network never reached the same level of correct output words, although it was trained five times longer than the hierarchical one.

To be most effective, the division into submodules should take place in a natural way, based on the properties of the task. This makes it *possible to change the function of the system in a modular fashion.* If the intermediate representation is meaningful, it is likely that additional modules can be added to the system with little modification, using the intermediate level as input or output.

In reading and generating stories, the sentence level provides a natural subgoal. The story networks of DISPAR contain the general semantic and script knowledge needed for inferencing, whereas the sentence networks form the specific language interface. Question-answering modules can be very easily added to the system, connecting them with the existing language-interface modules (Miikkulainen, 1990a). After a story is read into the internal representation, the only information that is actually stored are the role bindings. The knowledge about the events of the script is in the weights of the sentence- and story-generator networks. Different networks can be trained to paraphrase the story from the same role bindings in a different style or detail, *even in a different language.*

Modular training requires that training data must be provided for each module in the system. This is the cost of making a complex task tractable by splitting it up into modules. The reduction in complexity comes from injecting more knowledge into the system in terms of intermediate representations. An open research question for the future is how such hierarchical training schemes could arise automatically, that is, how the intermediate training targets could be discovered and constructed from surface-level data.

11.2 The role of the central lexicon

Developing I/O representations in a central lexicon does not seem to incur an extra cost on the training time. The lexicon of the hierarchical script-parser system (Section 11.1) was fixed at its final state, and the system was trained again from random initial weights without FGREP. Learning was faster at first, apparently because meaningful representations made the task easier. But the fine tuning of performance took longer because the system could not modify the representations to its advantage. The overall learning time turned out to be about the same as with FGREP.

Adding more modules to the system, all developing the same lexicon, does not seem to make learning any harder either. Learning in two different systems was compared: (1) the full four-module DISPAR was trained on four workstations in parallel, and (2) the two parser modules of DISPAR were trained on two workstations in parallel. The parsing subsystem of full DISPAR reached a satisfactory level of performance 30% *faster* in wall-clock time than the mere parsing system, despite the fact that in full DISPAR there were two extra modules modifying the representations also. This is surprising because with more modules, the representations have to encode more processing requirements, which at the outset should slow down the convergence. Apparently, *in a system with more modules, the representations become descriptive faster, which speeds up the total learning.*

In the experiments discussed in this article, the lexicon consists of conceptual word representations: It is a lexicon in the traditional sense of the word. The lexicon could play the role of a more general symbol table, containing representations for hierarchically more complex structures as well. This is essential in modeling formation of new concepts. A reduced description (see, e.g, Pollack, 1988) could be formed for a complex structure, and placed in the lexicon. A reference to the structure could be made using this lexicon entry, and communicated between modules like a word. A first step in this direction has already been taken in the DISPAR system. The internal representation for a story contains script and track slots which are filled with distributed representations of the different script and track types. These representations stand for higher order structures (coding information about specific roles and event order), although they are processed like words by the system.

The fact that FGREP can code both meanings of, for example, `bat` into a single representation is a good demonstration of the power of the mechanism. However, the distinction between the lexical word `bat` and the two concepts associated with it, `baseball-bat` and `live-bat` becomes blurred in this case. It would make more sense to develop separate representations for the distinct meanings of homonymous words.

A modified version of FGREP was developed to do this (Fellenz, 1989). The lexicon contains separate representations for the different meanings of ambiguous words, and the correct meaning is specified in the training data for each sentence. When an ambiguous word (e.g., `bat`) is encountered in the text, an *average* of the alternative concept representations (`baseball-bat` and `live-bat`) is input to the network. The network is trained to shoot for the correct meaning (e.g., `baseball-bat`) at the output. During backpropagation, the error signal is formed at the input layer as usual, but only the representation for the correct meaning is updated in the lexicon. The modified FGREP network develops representations for all different meanings of ambiguous words and learns to disambiguate among them whenever possible.

It is possible to go even further and completely separate the lexical and conceptual representations in the lexicon (Miikkulainen, 1990a). The actual input to the system consists of representations for the lexical items. The lexicon translates each lexical representation to the appropriate concept representation, and the system internally processes only concept representations.

11.3 Extending the lexicon

The system could extend its lexicon dynamically when needed. Each time a new word is encountered in the training data, the training supervisor would create a new entry for it in the lexicon. Expectations generated by the network could provide an initial guess for the representation. The expectation pattern in the appropriate slot (specified by the training data) is first matched against the lexicon. If there are no words with a similar representation, the expectation itself could be used as the initial pattern for the word. If there is a prototype close to the expectation pattern, a new instance of that prototype would be created and assigned to the new word. Because the expectation for `MaMaison` (as a new word) would probably be very much like `FANCY-RESTAURANT`, the initial lexicon entry for `MaMaison` would be an instance of `FANCY-RESTAURANT`.

The initial entry for the new word would be modified through experience, and it would acquire semantic content of its own. The instance would gradually become a new prototype itself. Eventually, there would be several different fancy-restaurant names in the vocabulary, each with an established unique meaning. The original `FANCY-RESTAURANT` would still be in the lexicon, representing a generalized form of these specific instances. This mechanism of learning new words could model forming of subclass hierarchies (i.e., more specific terms from general terms).

11.4 Implementing control with networks

In our simulations, an external symbolic system took care of the control of execution. Nothing very complicated is involved in it, and control could well be implemented with simple networks. In the performance phase of DISPAR, input and output segmentation is based on the period at the end of each sentence. Special networks could be trained to recognize the period, and control the gateways with multiplicative connections (Pollack, 1987; Rumelhart et al., 1986a). As soon as the sentence parser has read the period, one such gating network could open the pathway from the output of the sentence parser to the input of the story parser, which can then execute one output cycle. Similarly, another gating network would recognize the period at the output of the sentence generator, and send a signal back to the story generator. This signal would open the pathway from the hidden layer to the previous hidden layer, allowing the network to produce the next case-role representation (for an example of a similar connectionist control scheme, see Jain, 1989).

More generally, control could be implemented as an additional module, trained in the control task like any other module in the system. The control module would receive input from several pathways in the system, and its output would gate the pathways through multiplicative connections. This module could also monitor the performance of the system, detecting and correcting errors that are not automatically filtered out by the processing modules. It could monitor the feasibility of the task and the quality of output, and initiate exception processing when the usual mechanisms fail (e.g., for unusual events and deviations from the script). Interesting issues in generalization, expectations, and robustness of control make this an important research direction.

12 Discussion

12.1 PDP versus symbolic natural language processing

The main goal of the experiments was to demonstrate that modular FGREP networks are a plausible approach for modeling high-level cognitive tasks such as story paraphrasing and script recognition. The main advantage of this approach is that processing is learned from examples. The architecture is not committed to any particular data or knowledge structure; the same system can learn to process a wide variety of inputs, depending on the training data. A network trained to paraphrase restaurant scripts can learn to paraphrase travel stories and shopping stories just as well.

This contrasts with the traditional symbolic artificial intelligence models, where the processing instructions must be handcrafted with particular data in mind. These symbolic models cannot learn from statistical properties of the data, they can only do what their creator explicitly programmed them to do. For example, symbolic expectations must be implemented as specific rules for specific situations (Dyer, 1983; Schank and Abelson, 1977). Generalization to previously unencountered inputs is possible only if there exists a rule that specifies how this is done. Representations of these rules and their application is often very complex. The explicit rule-based approach seems unnatural, given how immediate and low level such operations as expectation and generalization are for people.

In the neural network approach, expectations and generalizations emerge automatically from the distributed character of processing. Knowledge for this is *based on statistical properties of the training examples, extracted automatically during training.* The resulting knowledge structures do not have explicit representations. For example, a script exists in a neural network only as statistical correlations coded in the weights. Every input is automatically matched to every correlation *in parallel.* There is no all-or-none instantiation of a particular knowledge structure. The strongest, most probable correlations will dominate, depending on how well they match the input, but all of them are simultaneously active at all times.

On the other hand, most symbolic models go beyond simple script instantiations in modeling story understanding. Complex texts contain references to several scripts and multiple scripts can be active at the same time. Scripts are used to connect the pieces of the text together; they are tools in building a complete representation of the story. However, stories strictly based on straightforward script instantiations are not particularly interesting. Any reasonable story contains unusual events or events deviating from the script, and that is often what makes the story worth telling. Symbolic systems have been developed that deal with these issues (Alvarado et al., 1990; Dyer, 1983; Schank and Abelson, 1977), based on higher level knowledge structures and processes, such as goals, plans, affects, beliefs, and argument structures.

PDP systems do not yet attempt story understanding at this level. They are very good at dealing with regularities, but do not easily lend themselves to processing the unusual or the unexpected (see Section 12.2). As long as the required inferences are based on statistical regularities, they can be well modeled with PDP. It is quite possible to build connectionist systems that process stories with multiple scripts (see, e.g., Sharkey et al., 1986), or even several simultaneously active scripts. Deviations from the ordinary events are harder to implement. This would require a higher level monitoring process that recognizes deviations and builds separate representations for them, possibly using techniques like plan analysis (Dolan, 1989; Lee, 1991). However, planning requires "dynamic inferencing" (Touretzky, 1991), that is, putting several pieces of information together to form genuinely novel information, not just pattern transformation. Dynamic inferencing is currently an open problem in PDP research.

12.2 Dealing with exceptions and novel situations

St. John (1990) studied the limitations of the PDP approach in processing unusual and novel situations in his story-gestalt model. Although his architecture is quite different – the story-gestalt model is an application of the sentence-gestalt architecture (St. John and McClelland, 1990) to sequences of propositions – his findings are very similar to what we have come across in DISPAR. Processing knowledge in these models is based on statistical regularities, and the models cannot handle deviations from the regularities very well. For example, if DISPAR reads a fancy-restaurant story where the food is bad, it will override the input and paraphrase the story with taste=`good`. This is because in all training examples the food in a fancy restaurant was good. The network learns this fact as part of the track, not as a role binding.

However, if the correlation between fancy restaurant and good food is broken with counterexamples in the training, the system has to make the taste of food a variable. This way, it is possible to train the network to handle deviations to a limited extent (St. John, 1990). For example, if in 90% of the fancy-restaurant stories the food was good, and in 10% it was bad, the network would develop a strong expectation for good food. After reading a sentence specifying that the food was actually bad, it would change the binding to bad. The overall behavior in this case resembles garden-path processing, but there is a significant difference. The network does not commit itself to any particular interpretation and then backtrack. All possibilities are kept active at all times, only their relative strengths change.

Training a network to handle exceptional inputs is very expensive. For a long time during training, the overall error goes down faster by simply ignoring the exceptional cases. Eventually the network has to pay more attention to them and it learns to process them correctly. The less frequent the exception, the harder it is to learn.

PDP systems generalize fairly well to novel situations that are in line with their training. They are good at interpolating within the training data, but they cannot extrapolate outside it very much. For example, having seen `The man ate the lobster`, the network might be able to process `The dog ate the lobster`, if both `man` and `dog` shared animate qualities in the data. A truly novel role binding, such as `The train ate the lobster`, which goes against all regularities in the training data, would be beyond its capability. The network has no reason to abstract the idea of a symbolic all-or-none role binding. The bindings only emerge from the statistical correlations of the constituents (St. John, 1990). As a result, PDP systems can naturally model semantic illusions, where the actual content of the text is overridden by semantically more likely content. But they cannot process truly novel role bindings according to a symbolic higher level rule. The two approaches are incompatible, as was pointed out by van Gelder (1989).

12.3 Processing types and tokens

The ID+content mechanism is a first step towards grounding symbolic reasoning in PDP. Any symbolic system needs to be able to deal with two kinds of information: (1) semantics of symbols, that is, knowledge about the properties of symbols and the relationships between them; and (2) identities of symbols, based on some unique surface-level tag such as a sensory perception of the referent (see, also, Harnad, 1990). The semantic knowledge is necessary for guiding the processing in a meaningful way, and the identities are necessary for logical reasoning and manipulation.

For example, suppose the system has the semantic knowledge that for some class of objects C, if $A \in C$, $property_1(A) \wedge property_2(A) \Rightarrow property_3(A)$, and it knows the facts $property_1(A_1)$ for $A_1 \in C$ and $property_2(A_2)$ for $A_2 \in C$. Even if this knowledge is statistical (i.e., is not exact but is true with a high probability), in order to draw the conclusion, the system must be able to verify that, in fact, $A_1 = A_2$ exactly, not just that they have similar statistical properties.

A common approach to symbolic modeling is to explicitly separate the two kinds of information. The semantic knowledge is maintained in "types," and each symbol is created as a "token," an instance of some type. The types form semantic hierarchies, the instances inherit the properties of parent types and also accumulate specific properties of their own during processing. Whether implemented in a symbolic semantic network (Quillian, 1967) or in a semantic network/PDP hybrid (Sumida and Dyer, 1989), in effect, there are two separate systems with a very complex interaction.

In the ID+content approach the identity and semantic content are *kept together in a single representation* and processed through the same pathways and structures. This is essential in building modular systems, because it allows modules to be connected with simple pathways. All the information is included in the output layer, and can be directly used as input in another module. Part of the representation is treated as the logical ID, and the rest is interpreted as the semantic content. The system is trained to process both parts simultaneously.

However, it turns out that processing identities is much harder than processing content. In the script-paraphrasing experiment, 90% of the training time was expended on the IDs. Fortunately, this appears to be a constant factor across the experiments, and not, for example, exponentially growing with the complexity of the task. Processing the IDs is hard because the rest of the input pattern provides no cue about what the IDs should be at the output. To produce a correct ID pattern, the network must copy it from the input exactly as it is, without any help from the context.

An interesting comparison can be made to human learning of logical thought (Inhelder and Piaget, 1958; Osherson, 1974). It seems that children pick up statistical semantics very fast, but are incapable of simple logical inference for a long time. Attributing properties only to specific instances seems to be a hard task for young children.

Maintaining a single, fixed ID for each object cannot be but a first crude approximation of actual identity. In reality, there are several different levels of identities for each object, varying in time and scope. For example, the person `John` today is not exactly the same as `John` 2 years ago, even though at another level `John` is a unique person over time. What is an appropriate identity depends on the processing context, and the actual boundary between ID and content is less well defined and much more complex than in DISPAR.

12.4 Parallel distributed inference

Inference is often modeled in artificial intelligence systems based on probabilistic formalisms (Buchanan and Shortliffe, 1985; Duda et al., 1977; Pearl, 1988). Events are known with certain probabilities and they provide evidence for other events. The dependencies are well defined, and inferences are drawn using conditional probabilities. Sound estimates of the certainty of the inferences are obtained, and coherent belief systems can be maintained. Unfortunately, this level of accuracy and rigor is very hard to achieve in high-level cognitive modeling, as in natural language understanding. The conditional probabilities are not well defined and dependencies are very complex and fuzzy. *Finding the relevant data is the main problem.* On the other hand, obtaining accurate measures of the reliability of the inference is not crucial. The task is to build a plausible hypothesis based on the most relevant data found in an enormous collection of information, knowing that this process is only an approximation of the best inference that could be drawn mathematically.

It seems that people have two fundamentally different mechanisms at their disposal for inferencing. The relevant data can be searched for using a sequential symbolic strategy. One does not have an immediate answer to the question, but the answer is sequentially constructed from stored knowledge by a high-level goal-directed process (e.g. by reasoning). Another type of inference occurs through associations immediately, in parallel, and without conscious control (i.e., by intuition). Large amounts of information, which may be incomplete or even conflicting, are simultaneously brought together to produce the most likely answer.

Neural network systems fit well into modeling intuitive inference (see, also, Touretzky, 1990). The network extracts statistical knowledge from the input data, and these statistics are brought together to generate the inference. For example, the amount of the tip in the restaurant story is inferred from the whole story, not just by looking at some part and applying a specific rule. If the food was bad, the network usually infers that the tip was small, but if the customer ate a hamburger, the representation in the tip slot will be closer to no-tip, because hamburgers are usually eaten in fast-food restaurants. In other words, neural networks are able to perform probabilistic inference, not by coming up with a specific answer and its probability, but by producing an answer that is the average of all alternatives, weighted by their probabilities. This is exactly what is needed in, for example, filling in the missing events and fillers in a script-based story. On the other hand, this type of inference is not useful when trying to understand unusual events (Section 12.2).

13 Conclusion

An approach to connectionist natural language processing is proposed which is based on hierarchically organized modular subnetworks and a central lexicon. Issues related to modularity are central in this work. Techniques are proposed for efficiently decomposing the task into modules, designing modular building blocks, establishing communication among modules, and modular training. We also show how high-level phenomena such as script-based inference, plausible role bindings, expectations, and learning of word semantics automatically emerge from the architecture.

The scale-up properties of the approach seem very good. Hierarchical modular structure with sequential communication efficiently reduces the complexity of the task. The modules filter out most of each other's errors rather than accumulate them. With meaningful intermediate representations it is possible to add more modules to the existing system, and these additions do not slow down the learning. It is possible to bootstrap the system with a small vocabulary, quickly increase its processing capability with the ID+content technique, and gradually learn more accurate processing. For these reasons, we believe that building from modular PDP networks with a central lexicon is a promising approach to modeling higher level natural language processing with distributed neural networks.

References

Alvarado, S., Dyer, M. G., and Flowers, M. (1990). Argument comprehension and retrieval for editorial text. *Knowledge-Based Systems*, 3(2):87–107.

Ballard, D. H. (1987). Modular learning in neural networks. In *Proceedings of the Seventh National Conference on Artificial Intelligence.* San Mateo, CA: Morgan Kaufmann.

Bower, G. H., Black, J. B., and Turner, T. J. (1979). Scripts in memory for text. *Cognitive Psychology*, 11:177–220.

Brousse, O. and Smolensky, P. (1989). Virtual memories and massive generalization in connectionist combinatorial learning. In *Proceedings of the 11th Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: Erlbaum.

Buchanan, B. G. and Shortliffe, E. H., editors (1985). *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Reading, MA: Addison-Wesley.

Cullingford, R. E. (1978). *Script Application: Computer Understanding of Newspaper Stories.* PhD thesis, New Haven, CT: Department of Computer Science, Yale University. Technical Report 116.

DeJong, G. F. (1979). *Skimming Stories in Real Time: An Experiment in Integrated Understanding.* PhD thesis, New Haven, CT: Department of Computer Science, Yale University. Technical Report 158.

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

Dolan, C. P. and Smolensky, P. (1989). Implementing a connectionist production system using tensor products. In Touretzky, D. S., Hinton, G. E., and Sejnowski, T. J., editors, *Proceedings of the 1988 Connectionist Models Summer School.* San Mateo, CA: Morgan Kaufmann.

Duda, R. O., Hart, P. E., Nilsson, N. J., Reboh, R., Slocum, J., and Sutherland, G. L. (1977). Development of a computer-based consultant for mineral exploration. Annual report, projects 5821 and 6415, Menlo Park, CA: SRI International.

Dyer, M. G. (1983). *In-Depth Understanding: A Computer Model of Integrated Processing for Narrative Comprehension.* Cambridge, MA: MIT Press.

Dyer, M. G. (1991). Symbolic NeuroEngineering for natural language processing: A multilevel research approach. In Barnden, J. and Pollack, J., editors, *Advances in Connectionist and Neural Computation Theory, Vol. 1: High Level Connectionist Models.* Norwood, NJ: Ablex.

Dyer, M. G., Cullingford, R. E., and Alvarado, S. (1987). Scripts. In Shapiro, S. C., editor, *Encyclopedia of Artificial Intelligence.* New York: Wiley.

Elman, J. L. (1989). Structured representations and connectionist models. In *Proceedings of the 11th Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: Erlbaum.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2):179–211.

Feldman, J. A. (1989). Neural representation of conceptual knowledge. In Nadel, L., Cooper, L. A., Culicover, P., and Harnish, R. M., editors, *Neural Connections, Mental Computation.* Cambridge, MA: MIT Press.

Fellenz, C. B. (1989). A connectionist model of linguistic analysis. Master's thesis, Chico, CA: California State University, Chico.

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

Harnad, S. (1990). The symbol grounding problem. *Physica D*, 42:335–346.

Harris, C. L. and Elman, J. L. (1989). Representing variable information with simple recurrent networks. In *Proceedings of the 11th Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: Erlbaum.

Hartigan, J. A. (1975). *Clustering Algorithms.* New York: Wiley.

Hinton, G. E. (1981). Implementing semantic networks in parallel hardware. In Hinton, G. E. and Anderson, J. A., editors, *Parallel Models of Associative Memory.* Hillsdale, NJ: Erlbaum.

Hinton, G. E. (1986). Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: Erlbaum.

Inhelder, B. and Piaget, J. (1958). *The Growth of Logical Thinking from Childhood to Adolescence.* New York: Basic Books.

Jacobs, R. A. (1990). *Task Decomposition Through Competition in a Modular Connectionist Architecture.* PhD thesis, Amherst, MA: Department of Computer and Information Science, University of Massachusetts, Amherst.

Jain, A. N. (1989). A connectionist architecture for sequential symbolic domains. Technical Report CMU-CS-89-187, Pittsburgh, PA: Computer Science Department, Carnegie Mellon University.

Jordan, M. I. (1986). Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: Erlbaum.

Kohonen, T. (1984). *Self-Organization and Associative Memory.* Berlin; Heidelberg; New York: Springer.

Korf, R. E. (1987). Planning as search: A quantitative approach. *Artificial Intelligence*, 33(1):65–88.

Lee, G. (1991). *Distributed Semantic Representations for Goal/Plan Analysis of Narratives in a Connectionist Architecture.* PhD thesis, Los Angeles: Computer Science Department, University of California, Los Angeles.

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.* Cambridge, MA: MIT Press.

McClelland, J. L., Rumelhart, D. E., and Hinton, G. E. (1986). The appeal of parallel distributed processing. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations.* Cambridge, MA: MIT Press.

Miikkulainen, R. (1990a). *DISCERN: A Distributed Artificial Neural Network Model of Script Processing and Memory.* PhD thesis, Los Angeles: Computer Science Department, University of California, Los Angeles.

Miikkulainen, R. (1990b). A distributed feature map model of the lexicon. In *Proceedings of the 12th Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: Erlbaum.

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

Miikkulainen, R. (1990d). Script recognition with hierarchical feature maps. *Connection Science*, 2(1&2):83–101.

Miikkulainen, R. and Dyer, M. G. (1987). Building distributed representations without microfeatures. Technical Report UCLA-AI-87-17, Los Angeles: Computer Science Department, University of California, Los Angeles.

Miikkulainen, R. and Dyer, M. G. (1988). Forming global representations with extended backpropagation. In *Proceedings of the IEEE International Conference on Neural Networks.* Piscataway, NJ: IEEE.

Miikkulainen, R. and Dyer, M. G. (1989a). 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.* San Mateo, CA: Morgan Kaufmann.

Miikkulainen, R. and Dyer, M. G. (1989b). A modular neural network architecture for sequential paraphrasing of script-based stories. In *Proceedings of the International Joint Conference on Neural Networks.* Piscataway, NJ: IEEE.

Minsky, M. (1963). Steps toward artificial intelligence. In Feigenbaum, E. A. and Feldman, J. A., editors, *Computers and Thought.* New York: McGraw-Hill.

Minsky, M. (1985). *Society of Mind.* New York: Simon & Schuster.

Osherson, D. N. (1974). *Logical Abilities in Children.* Hillsdale, NJ: Erlbaum.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* San Mateo, CA: Morgan Kaufmann.

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

Pollack, J. B. (1988). Recursive auto-associative memory: Devising compositional distributed representations. In *Proceedings of the 10th Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: Erlbaum.

Quillian, M. R. (1967). Word concepts: A theory and simulation of some basic semantic capabilities. *Behavioral Science*, 12:410–430.

Ritter, H. (1989). Combining self-organizing maps. In *Proceedings of the International Joint Conference on Neural Networks.* Piscataway, NJ: IEEE.

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.* 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.* Cambridge, MA: MIT Press.

Rumelhart, D. E. and McClelland, J. L. (1987). Learning the past tenses of English verbs: Implicit rules or parallel distributed processing. In MacWhinney, B., editor, *Mechanisms of Language Acquisition.* Hillsdale, NJ: Erlbaum.

Schank, R. and Abelson, R. (1977). *Scripts, Plans, Goals, and Understanding - An Inquiry into Human Knowledge Structures.* Hillsdale, NJ: Erlbaum.

Schank, R. and Riesbeck, C. K., editors (1981). *Inside Computer Understanding.* Hillsdale, NJ: Erlbaum.

Sejnowski, T. J. and Rosenberg, C. R. (1987). Parallel networks that learn to pronounce English text. *Complex Systems*, 1:145–168.

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*, volume 1. San Mateo, CA: Morgan Kaufmann.

Sharkey, N. E., Sutcliffe, R. F. E., and Wobcke, W. R. (1986). Mixing binary and continuous connection schemes for knowledge access. In *Proceedings of the Sixth National Conference on Artificial Intelligence.* San Mateo, CA: Morgan Kaufmann.

Simon, H. (1981). *The Sciences of the Artificial.* Cambridge, MA: MIT Press.

St. John, M. F. (1990). *The Story Gestalt – Text Comprehension by Cue-Based Constraint Satisfaction.* PhD thesis, Pittsburgh, PA: Department of Psychology, Carnegie Mellon University.

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

Sumida, R. A. and Dyer, M. G. (1989). Storing and generalizing multiple instances while maintaining knowledge-level parallelism. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence.* San Mateo, CA: Morgan Kaufmann.

Touretzky, D. S. (1991). Connectionism and compositional semantics. In Barnden, J. and Pollack, J., editors, *Advances in Connectionist and Neural Computation Theory, Vol. 1: High Level Connectionist Models.* Norwood, NJ: Ablex.

van Gelder, T. (1989). *Distributed Representation.* PhD thesis, Pittsburgh, PA: Department of Philosophy, University of Pittsburgh.

Waibel, A. (1989). Connectionist glue: Modular design of neural speech systems. In Touretzky, D. S., Hinton, G. E., and Sejnowski, T. J., editors, *Proceedings of the 1988 Connectionist Models Summer School.* San Mateo, CA: Morgan Kaufmann.

	Sentence Frame	Correct Case Roles
1.	The human ate.	agent
2.	The human ate the food.	agent, patient
3.	The human ate the food with the food.	agent, patient, modif
4.	The human ate the food with the utensil.	agent, patient, instr
5.	The animal ate.	agent
6.	The predator ate the prey.	agent, patient
7.	The human broke the fragileobj.	agent, patient
8.	The human broke the fragileobj with the breaker.	agent, patient,instr
9.	The breaker broke the fragileobj.	instr, patient
10.	The animal broke the fragileobj.	agent, patient
11.	The fragileobj broke.	patient
12.	The human hit the thing.	agent, patient
13.	The human hit the human with the possession.	agent, patient, modif
14.	The human hit the thing with the hitter.	agent, patient, instr
15.	The hitter hit the thing.	instr, patient
16.	The human moved.	agent, patient
17.	The human moved the object.	agent, patient
18.	The animal moved.	agent, patient
19.	The object moved.	patient

Table 5: **Sentence templates.** Adapted from McClelland and Kawamoto (1986, Table 3, p. 291) with permission.

Category	Nouns
human	man woman boy girl
animal	bat chicken dog sheep wolf lion
predator	wolf lion
prey	chicken sheep
food	chicken cheese pasta carrot
utensil	fork spoon
fragileobj	plate window vase
hitter	bat ball hatchet hammer vase paperwt rock
breaker	bat ball hatchet hammer paperwt rock
possession	bat ball hatchet hammer vase dog doll
object	bat ball hatchet hammer paperwt rock vase plate window fork spoon pasta cheese chicken carrot desk doll curtain
thing	human animal object

Table 6: **Noun categories.** Adapted from McClelland and Kawamoto (1986, Table 2, p. 290) with permission.

APPENDIX A: Sentence data

The sentence templates are listed in Table 5. Each frame has one to three noun slots. Each slot has a predetermined case role, shown at right. Each slot can be filled with any of the nouns in the specified category, listed in Table 6. For instance, The human ate the food generates 4×4 different sentences, all with the case-role assignment human=agent, food=patient.

APPENDIX B: Story data

The templates for each track are listed below. The slots (i.e., roles) are the same for all tracks within a script, whereas the fillers vary from track to track. Words in upper case are prototype words. Actual stories are formed from these templates by specifying the ID part for each prototype word. For example, in the example fancy-restaurant story (Section 9, Figures 12 and 13) the prototype word PERSON has been instantiated with `John`, FANCY-FOOD with `lobster` and FANCY-REST with `MaMaison`, each with the ID pattern (1 0). The incomplete stories for performance tests consisted of sentences marked with "*".

Sentences were represented with six case-role assemblies: agent, act, recipient, patient-attribute, patient, and location. Certain related case roles, such as from-, to- and at-location, were actually represented by the same assembly for simplicity. The interpretation of the location assembly (among others) depends on the representation in the act assembly. In other words, the sentence representation is data-specific.

RESTAURANT SCRIPT

Slots: script, track, customer, food, restaurant, taste, tip

Fancy-restaurant track

Fillers: `$restaurant, $fancy, PERSON, FANCY-FOOD, FANCY-REST, good, big/small`

```
PERSON went to FANCY-REST.*
The waiter seated PERSON.
PERSON asked the waiter for FANCY-FOOD.*
PERSON ate a good FANCY-FOOD.
PERSON paid the waiter.
PERSON left a big/small tip.*
PERSON left FANCY-REST.
```

Coffee-shop-restaurant track

Fillers: `$restaurant, $coffee, PERSON, COFFEE-FOOD, COFFEE-REST, good/bad, big/small`

```
PERSON went to COFFEE-REST.*
PERSON seated PERSON.
PERSON asked the waiter for COFFEE-FOOD.*
PERSON ate a good/bad COFFEE-FOOD.*
PERSON left a big/small tip.
PERSON paid the cashier.
PERSON left COFFEE-REST.
```

Fast-food-restaurant track

Fillers: `$restaurant, $fast, PERSON, FAST-FOOD, FAST-REST, bad, none`

```
PERSON went to FAST-REST.*
PERSON asked the cashier for FAST-FOOD.*
PERSON paid the cashier.
PERSON seated PERSON.
PERSON ate the small FAST-FOOD.*
The FAST-FOOD tasted bad.
PERSON left FAST-REST.
```

SHOPPING SCRIPT

Slots: script, track, customer, item, store

Clothing-shopping track

Fillers: `$shopping, $clothing, PERSON, CLOTH-ITEM, CLOTH-STORE`

```
PERSON went to CLOTH-STORE.*
PERSON looked for good CLOTH-ITEM.*
PERSON tried on several CLOTH-ITEM.
PERSON took the best CLOTH-ITEM.*
PERSON paid the cashier.
PERSON left CLOTH-STORE.
```

Electronics-shopping track

Fillers: `$shopping, $electronics, PERSON, ELECTR-ITEM, ELECTR-STORE`

```
PERSON went to ELECTR-STORE.*
PERSON looked for good ELECTR-ITEM.*
PERSON asked the staff questions about ELECTR-ITEM.
PERSON took the best ELECTR-ITEM.*
PERSON paid the cashier.
PERSON left ELECTR-STORE.
```

Grocery-shopping track

Fillers: `$shopping, $grocery, PERSON, GROCERY-ITEM, GROCERY-STORE`

```
PERSON went to GROCERY-STORE.*
PERSON took a big shopping-cart.
PERSON compared GROCERY-ITEM prices.*
PERSON took several GROCERY-ITEM.*
PERSON waited in a big line.
PERSON paid the cashier.
PERSON left GROCERY-STORE.
```

TRAVEL SCRIPT

Slots: script, track, traveler, origin, destination, distance

Plane-travel track

Fillers: `$travel, $plane, PERSON, PLANE-ORIGIN, PLANE-DEST, big`

```
PERSON went to PLANE-ORIGIN.*
PERSON checked-in for a flight to PLANE-DEST.*
PERSON waited at the gate for boarding.
PERSON got-on the plane.
The plane took-off from PLANE-ORIGIN.
The plane arrived at PLANE-DEST.*
PERSON got-off the plane.
```

Train-travel track

Fillers: `$travel,`