

**Computer Science Department Technical Report
Artificial Intelligence Laboratory
University of California
Los Angeles, CA 90024-1596**

**NATURAL LANGUAGE PROCESSING WITH MODULAR
NEURAL NETWORKS AND DISTRIBUTED LEXICON**

**Risto Miikkulainen
Michael Dyer**

**January 1990
CSD-900001**

NATURAL LANGUAGE PROCESSING WITH MODULAR NEURAL NETWORKS AND DISTRIBUTED LEXICON *†

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

Abstract

An architecture for connectionist natural language processing is presented, which is based on hierarchically organized modular subnetworks together with a central lexicon of distributed input/output representations. Using the FGREP method, the backpropagation error signal is utilized to develop the representations automatically while the system is learning the processing task. The representations end up reflecting the properties of the input items which are most crucial to the task, facilitating excellent generalization and expectations about possible contexts. The lexicon can be extended by cloning new instances of the items, i.e. generating a number of items with the same properties but with distinct identities. 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 modules is trained to produce fully expanded paraphrases of script-based stories, where unmentioned events and role fillers are inferred.

1 Introduction

Connectionist models have recently emerged as an alternative to symbolic modeling in cognitive science. Their major appeal is that processing can be learned from examples, based on statistical regularities in the data. 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].

The models typically have very little internal structure. They produce the statistically most likely answer given the input conditions, in a process which is opaque to the external observer. This suits well into modeling well-defined, isolated low-level tasks, such as learning past tense

*This research was supported in part by a grant from the 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 San Diego Supercomputer Center, and on equipment donated to UCLA by Hewlett Packard.

†Submitted to *Cognitive Science*.

forms or pronunciation of words. A plausible approach for higher-level cognitive modeling would be to compose the model from several simple submodules, which work together to produce the higher-level behaviour [Minsky, 1985]. Some of the central issues in this approach are how the interactions between the modules are organized and how communication between modules is accomplished.

The modules need to have a common set of terms, a common language to effectively work together. In a large system consisting of many modules, with many communicating pairs, the most efficient way to establish this is through a global vocabulary, a central lexicon. Communication between each pair of modules is established by using the terms from this global symbol table, instead of having a separate set of terms for each communication channel.

In a connectionist high-level model, the communication (i.e. input/output of each module) could take place using distributed representations. This is a major advantage of the connectionist approach in general. Distributed representations can reflect the meanings of the concepts they stand for. Similar concepts have similar representations, which results in several interesting processing enhancements. Neural network modules can perform their task in less than perfect conditions, e.g. under noise or damage, and generalize their processing knowledge into previously unseen situations and perform reasonably well even when the input is incomplete or somewhat conflicting.

One way to establish a central, global table of meaningful distributed representations is to compose them beforehand from a predefined set of microfeatures [Hinton, 1981; McClelland and Kawamoto, 1986]. The microfeatures are chosen relevant to the task and the coding is a way to inject external knowledge into the system. Alternatively, the representations can be developed automatically by the system itself while it is learning the processing task. These representations reflect the regularities in all subtasks, extracted without external supervision [Miikkulainen and Dyer, 1988; Miikkulainen and Dyer, 1989a].

This article examines the properties of global distributed representations of the latter kind, and the prospects of building models of complex cognitive systems based on modules communicating with these representations. We concentrate on natural language processing tasks, where the lexicon consists of distributed representations of words. The mechanism of forming global representations with extended backpropagation (FGREP) is first introduced in

the context of sentence processing, i.e. in the task of assigning roles to sentence constituents. We also show how the lexicon can be extended by creating several distinct words with the same properties (e.g. John, Mary, Bill from the word human).

The FGREP-method is then extended to sequential input. The word representations are developed in the task of mapping a sequence of input words into the case-role representation of the sentence. Finally, we show how an order of magnitude more complex system can be built from hierarchically organized recurrent FGREP modules, together with a central lexicon of words. This system learns to produce fully expanded paraphrases of script-based input stories, where unmentioned events are inferred, and unspecified fillers are inferred. Discussion of the general plausibility of the approach and future prospects concludes the paper.

2 Methods for forming distributed representations

Sentence case-role assignment is an example of a cognitive task which is well suited for modelling 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, the input/output items are represented as different patterns of activity over the same set of units. Desirable properties achieved are: (1) it is possible to associate similar items and generalize properties by sharing the same activity subpatterns, and (2) the system is robust against noise and damage [Hinton *et al.*, 1986].

One approach for forming distributed representation patterns is **semantic microfeature encoding**, used e.g. by McClelland and Kawamoto in the case-role assignment task [McClelland and Kawamoto, 1986] (see also [Hinton, 1981]). Each concept is classified along a predetermined set of dimensions such as human-nonhuman, soft-hard, male-female etc. Each microfeature 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 the assembly of units (figure 1).

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

On the other hand, the patterns must be pre-encoded and they remain fixed. Performance cannot be optimized by adapting the representations to 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 advantageous to use is a hard problem. There is also the

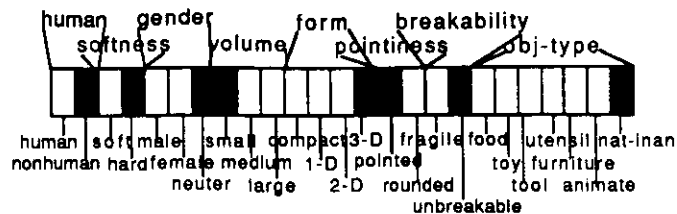


Figure 1: **Semantic microfeature encoding of the word rock** (after [McClelland and Kawamoto, 1986]). A group of units is assigned to each semantic microfeature. The units stand for different values (bottom of figure) of the semantic dimensions (top). Black unit indicates that the value is on, white off.

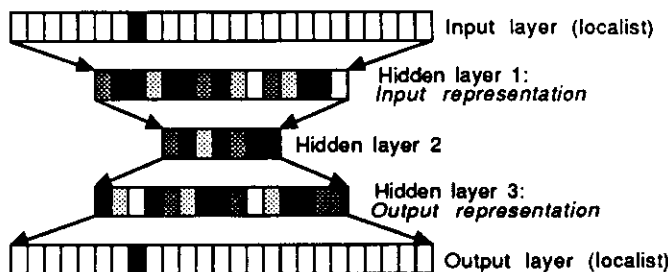


Figure 2: **Developing internal representations in hidden layers.** The activity of a single unit in the input layer is propagated through the input weights to the first hidden layer. The resulting pattern (indicated as grey-scale values) constitutes the input representation for this input item. The pattern in the third hidden layer, which results in a single unit being turned on in the output layer, is the output representation for that item. These representations are in general different even if the input and output items are the same.

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 (see e.g. the family tree example in [Hinton, 1986]). A network of this type usually consists of input, output and three hidden layers (figure 2). The input and output layers are localist, i.e. 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/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].

This approach does not address the issue of encoding input/output representations. The system does not deal with the representations per se; they develop as a side effect of modifying the weights to improve the task performance. The patterns are not available outside the net-

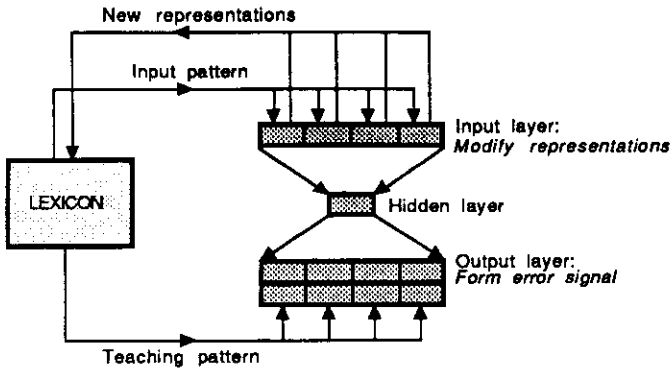


Figure 3: **Basic FGREP architecture.** The system consists of a three-layer backpropagation network and an external, global lexicon containing the input/output representations. At the end of each backpropagation cycle, the current input representations are modified at the input layer according to the error signal. The new representations are loaded back to the lexicon, replacing the old ones.

work, and they are not used in communication with the network. Moreover, since both penultimate layers develop their activity patterns independently, each item has two different representations: one as an input and another one as an output item. These activity patterns are *local, internal processing aids* more than input/output representations which can be used in a larger environment.

In the **FGREP** approach [Miikkulainen and Dyer, 1988; Miikkulainen and Dyer, 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/output to the network and they 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 backPropagation

3.1 Basic architecture

The FGREP mechanism is based on a basic three-layer backward error propagation network (figure 3). The network learns the processing task by adapting the connection weights according to the standard backpropagation equations [Rumelhart *et al.*, 1986b, pages 327-329]. At the same time, representations for the input data are developed at the input layer according to the error signal extended to the input layer. Input and output layers are divided into assemblies and several items are represented and modified simultaneously.

The representations are stored in an external lexicon network. A routing network forms each input pattern and the corresponding teaching pattern by concatenating the lex-

icon entries of the input and teaching items. Thus the same representation for each item is used in different parts of the backpropagation network, both in the input and in the output.

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

3.2 Extending backpropagation to the representations

Standard backpropagation produces an error signal for each hidden layer unit. By propagating the signal one layer further to the input layer, the representations can be changed as if they were weights on connections coming in to the input layer.

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 of an input unit is the identity function and its derivative is one. The error signal can now be computed for each input unit as a simple case of the general error signal equation [Rumelhart *et al.*, 1986b, Eq.14]:

$$\delta_{1i} = \sum_j \delta_{2j} w_{1ij}. \quad (1)$$

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.

Imagine a localist 0:th layer before the input layer, with one unit dedicated to each input item in each assembly. In this layer at most one unit per assembly is active with value 1 at any time (the one corresponding to the current input); the rest of the units have zero activity. Each localist unit is connected to all units in the input assembly with weights equal to the input representations. Extending the back propagation weight change to these weights can be interpreted as changing the representations themselves:

$$\Delta r_{ci} = \eta \delta_{1i} r_{ci}, \quad (2)$$

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. Using this analogy, *representation learning is implemented as an extension of the back propagation algorithm*. While the weight values are unlimited, 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

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

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

Note that the backpropagation “sees” the representations simply as an extra layer of weights. By separating the representations from the network and treating them as global, external objects (instead of local, internal weights) we can develop a single, concrete representation for each item. Since 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 which are most crucial to the task.*

3.3 Reactive training environment

The process differs from ordinary backpropagation in that both the input and the teaching 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 3). The activity is propagated through the network to the output layer, where the error signal is formed by comparing the output pattern to the teaching pattern, which is also formed by drawing the current representations from the lexicon. The error signal is propagated back to the input layer, changing weights and the input item representations along the way. Next time the *same input* occurs, the output will be closer to the *same teaching pattern*. The modified representations are now put back to the lexicon, replacing the old ones and thereby *changing the next teaching pattern for the same input*. The shape of the error surface is changing at each step, i.e. 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. The 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. The changes in the error surface are a form of noise (a noisy error signal), which backpropagation in general tolerates very well.

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 the 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 e.g. 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 (figure 4). This requires forming a shallow semantic interpretation of the sentence.

For example, in **The ball hit the girl with the**


INPUT:	Syntactic constituents	Subj. ball	Verb hit	Object girl	With dog	
						
OUTPUT:	Case-role assignment	Agent -	Act hit	Patient girl	Instr. ball	Modif. dog

Figure 4: Assigning case roles to sentence constituents. The example sentence is **The ball hit the girl with the dog.**

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. 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**.

In [McClelland and Kawamoto, 1986] the authors describe a system which learns to assign case roles to sentence constituents. The same task with the same data was used to test FGREP, partly because it provides a convenient comparison to a system using fixed microfeature encoding. The task was restricted to a small repertoire of sentences studied in the original experiment. The sentence generators are depicted in table 1 and the noun categories in table 2.

The sentence frames and the noun 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 teaching input is made up from the input sentence constituents (figure 5). This is by no means necessary for learning the representations. The required output of the network could be anything and the FGREP method would work the same. A discriminatory or “pigeonholing” task is actually harder than a general task because of the reactive training effect.

5 Properties of FGREP-representations

5.1 Simulations

The learning is fairly insensitive to the simulation parameters and the system configuration parameters. The on-line 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 ran-

Gen. 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 1: Sentence generators.

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 2: Noun categories.

The generators are presented as sentence frames, with one to three noun slots. Each slot can be filled with any of the nouns in the specified category (table 2), and each slot has a predetermined case role. For instance, *The human ate the food* generates 4×4 different sentences, all with the case-role assignment *human* = *agent*, *food* = *patient*.

dom order, and by gradually reducing the learning rate. The results reported in the following three sections are from the 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 the 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, the 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 representation 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 5 shows a snapshot of the simulation after a real-time display on an HP 9000/350 workstation. Of the 1475 sentences produced by the sentence generators, 1439 were used for training and 38 were reserved for testing.

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. The 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 6 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 the

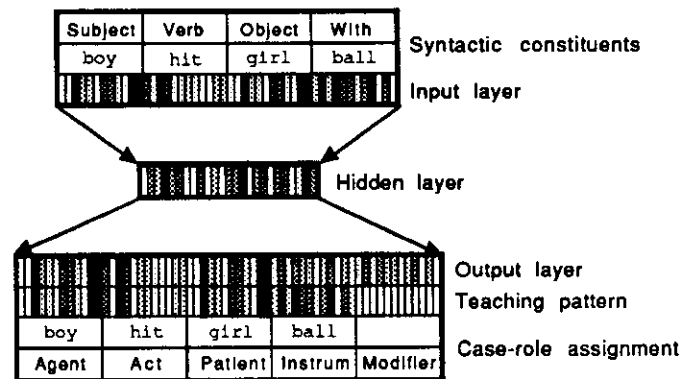


Figure 5: Snapshot of basic FGREP simulation. The input and output layers of the network are divided into assemblies, each of which holds one word representation at a time. Each unit in an input assembly is set to the activity value of the corresponding component in the lexicon entry. The input layer 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 which is identical to the lexicon representation of the word that fills that role. The correct role assignment is shown at the bottom of the display. This pattern forms the teaching input to the network. Grey-scale values from white to black are used in the figure to code the unit activities in the range $[0,1]$.

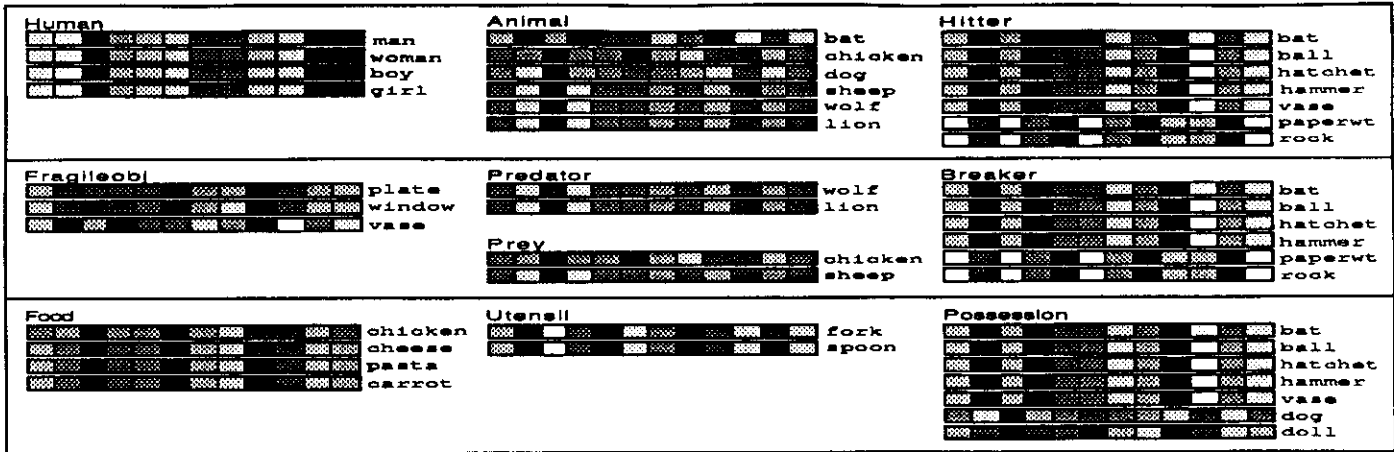


Figure 6: Final representations. The representations for the synonymous words {man, woman, boy, girl}, {fork, spoon}, {wolf, lion}, {plate, window}, {ball, hatchet, hammer}, {paperwt, rock} and {cheese, pasta, carrot} have become almost identical.

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. {man, woman, boy, girl} forms one such group, {fork, spoon}, {wolf, lion}, {plate, window}, {ball, hatchet, hammer}, {paperwt, rock} and {cheese, pasta, carrot} others. If there is at least one difference in the usage of two nouns, their representations become different. The discriminating input modifies one of the representations while the other one remains the same.

Since each noun belongs to several categories its representation can be seen as evolving from the competition between the categories. This is clearest on the part of the ambiguous nouns chicken and bat, which on one hand 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. 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 6 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, if one runs a merge clustering algorithm [Hartigan, 1975] on the representations, the optimal clusters turn out to be quite similar to the noun categories (figure 7).

Inspection of the representations in figure 6 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 8).

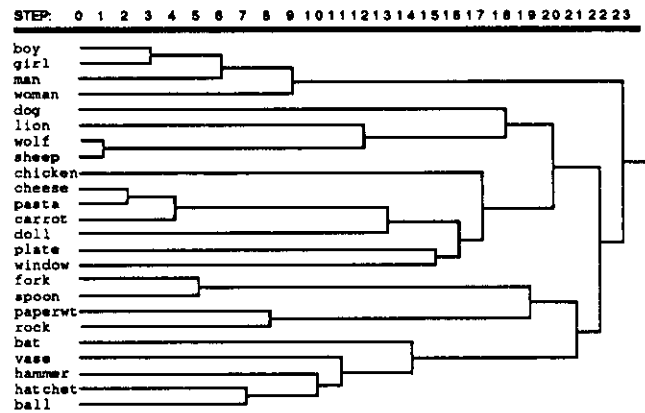


Figure 7: Merge clustering the representations. At each time step, the clusters with the shortest average Euclidian distance were merged.

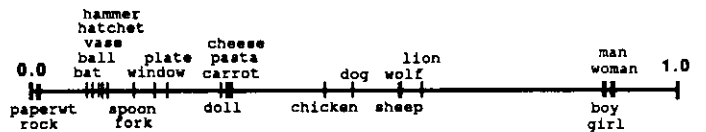


Figure 8: Categorization by unit 11. The words are placed on a continuous line [0,1] according to the value of the last unit in their representations.

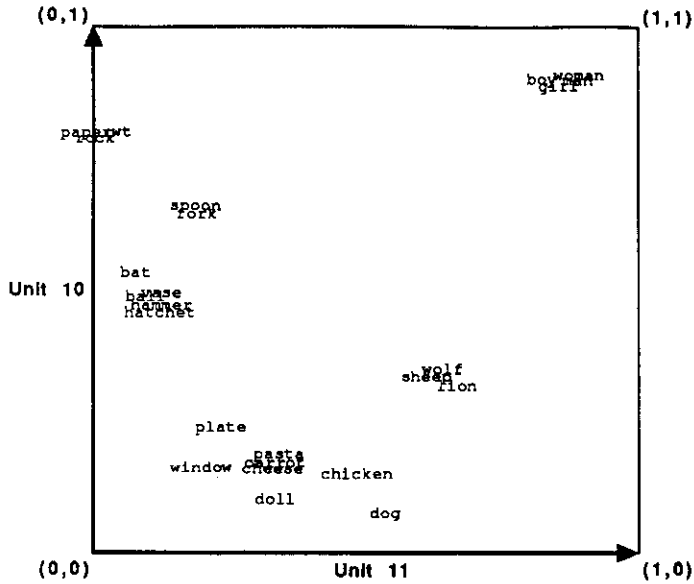


Figure 9: Categorization by units 10 and 11. The words are mapped on the unit square according to the values of the last two units in their representations.

Each unit provides a unique perspective into the words, by coding slightly different properties of the word space. Combining the values of two units thus provides an even more descriptive categorization (figure 9), and the complete representation can be seen as a combination of twelve 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, i.e. 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 10). Note that the ambiguous nouns *chicken* and *bat* are mapped between their two possible categories *animal* and *food* and *animal* and *hitter*, and also *vase* is mapped between *fragile-obj* and *hitter*.

It is very hard to name the properties which the individual units are actually coding. In [Hinton, 1986] the author was able to give interpretation for some of the units in the hidden layer representation, although he points out that the system develops its own microfeatures, which may or may not correspond to ones that humans would use to characterize data. Our results suggest that the microfeatures in the resulting representation in general are not identifiable. With complex input data, the individual units will become sensitive to a combination of several features which is very unlikely to match an established term.

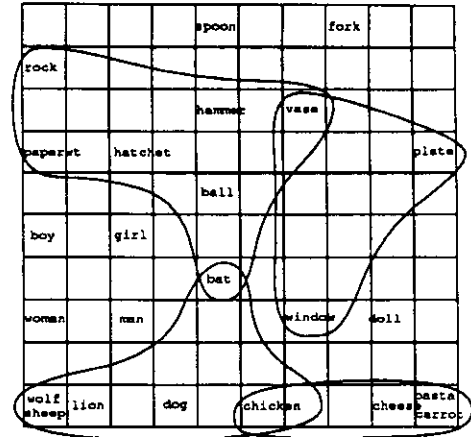


Figure 10: 2-D Kohonen-map of the representations. Labels indicate the maximally responding unit in the 10×10 feature map network for each representation vector. The map was formed in 15,000 epochs, where the neighborhood radius was decreased from 4 to 1 and the learning rate from 0.5 to 0.05 in the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

5.3 Performance in the task

The performance in the role assignment task was tested with two test sets. The first one consisted of two sentences from each generator which had been used in the training, and the second one consisted of the test sentences which the network had not seen before. Tables 3 and 4 show the results for each sentence.

The system learns the correct assignment of 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 5. *Ball* can be either 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*.

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 11 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 hidden-layer networks but also partly due to the fact that the representation is not coded into specific microfeature-units, but is distributed over all units in a holographic fashion. Eliminating a unit means removing

Gen. Input	$E_i < 0.15$	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 3: Performance, familiar sentences.

The leftmost entry in each row identifies the generator which produced the sentence (referring to table 1). The first figure after each sentence indicates the percentage of output units whose values were within 0.15 of the correct output value (which ranged from 0 to 1). The second figure indicates the average error per unit. Ambiguous sentences are marked with "?". The test sentence sets are identical to those used in [McClelland and Kawamoto, 1986].

one classification perspective, and these perspectives are apparently additive.

With a third of the input units removed, the system still gets 77% of the output within 0.15 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 still performs above chance level. Information about the input space distribution has been stored in the weights, and the network produces a best guess, i.e. an average of all possible outputs.

5.5 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. The generalization in FGREP has a different character. The network *never has to extend its processing into very unfamiliar patterns, because the generalization has already been done on the 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.

Gen. Input	$E_i < 0.15$	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 4: Performance, unfamiliar sentences.

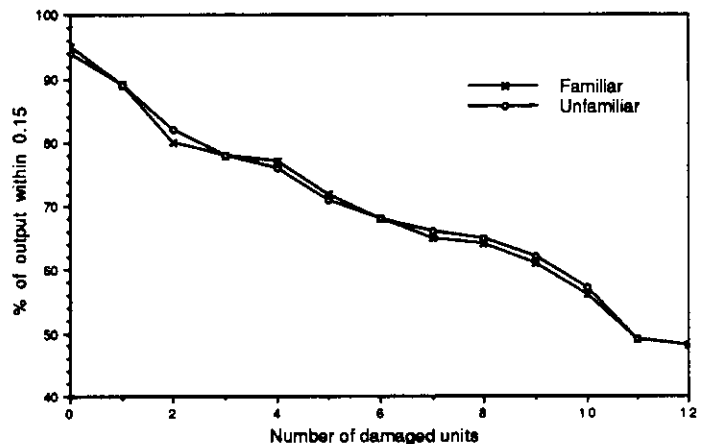


Figure 11: Damage resistance. The Familiar and Unfamiliar data sets are listed in tables 3 and 4. The horizontal axis indicates the number of units eliminated from the 12-unit representation, and the vertical axis indicates the percentage of output units which were within 0.15 of the correct value.

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**, since 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 training data. E.g. **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 generators (table 1). It does not belong to the input space the system is trying to learn, i.e. 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 3 and 4). This is in contrast to systems using representations with precoded, fixed microfeatures, such as McClelland and Kawamoto's. Even though two words are equivalent in the input data, their microfeature representations remain different, and the same level of generalization is much harder to achieve.

An interesting question is, how well FGREP can 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 number of sentences from each generator were selected randomly for the training, including all sentences in the Familiar set and excluding all in the Unfamiliar set. When a generator produced fewer sentences than needed, multiple copies were used.

The performance as a function of the training set size is plotted in figure 12. With very small training sets the system learns the idiosyncronies of the training sentences, and generalizes poorly. Generalization improves very fast as more training data is included. A critical mass is reached at around eight sentences per generator, which covers approximately 10% of the input space. At this point the system gets 96% of the unfamiliar sentence output within 0.15, and adding more training data does not significantly improve performance.

Even with the very incomplete training data, the final representations end up reflecting the similarities of the words very well. Only the fine tuning of the equivalent words is lost, simply because there are no equivalent words 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*

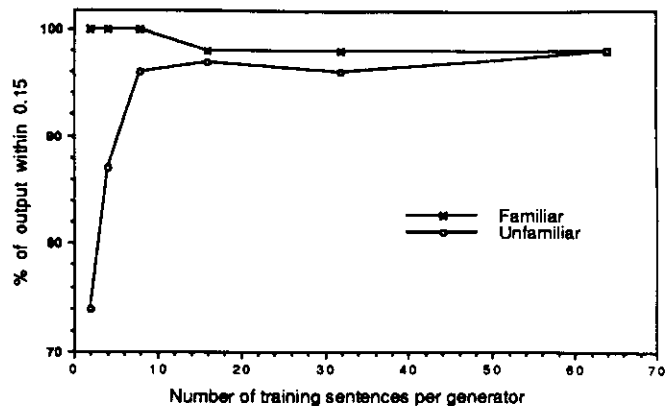


Figure 12: Performance as a function of the training set size. The Familiar and Unfamiliar data sets are listed in tables 3 and 4. The horizontal axis indicates the number of sentences per generator that were used in the training, and the vertical axis indicates the percentage of output units which were within 0.15 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 to 15,000 epochs, depending on size of the training set.

to the last 90% of the input space.

The results in figure 12 are actually better than those presented in tables 3 and 4, because the training data was selected differently. In the generalization experiment, the approach was to train the system *with all generators 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*. Since some generators produce 3 different sentences while 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 0.023, while 97% of the input units are within 0.15 (table 5). Both types of tests will be used in later sections.

5.6 Creating expectations about possible contexts

The whole pattern in the input and teaching layers has an effect on how each input item is modified during the back-propagation cycle. The context of an input item can therefore be defined operationally as the whole input-output 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 contexts*. The more frequent the context, the stronger is 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

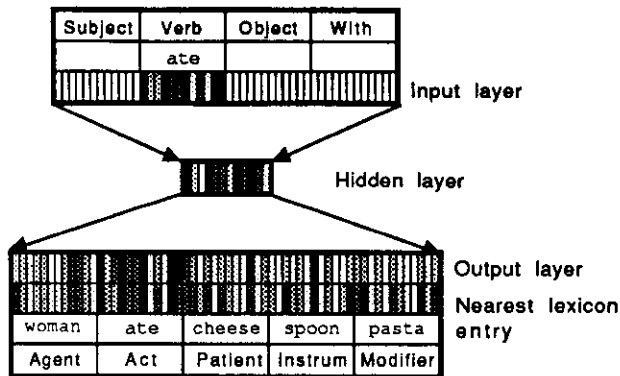


Figure 13: **Expectations embedded in the word ate.** The bottom layer show the lexicon entry which is closest to the output generated by the network (Euclidian distance of normalized vectors). The effect is more pronounced when a small hidden layer is used. In this example a network with 12 hidden units was trained for 50 epochs with a learning rate 0.1.

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 13). The resulting pattern at the output layer indicates the most likely context. As can be seen from the figure, 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. The distributed expectations could replace the symbolic expectations traditionally used in natural language conceptual analyzers, e.g. [Dyer, 1983].

6 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.

However, knowledge about the use of the words alone cannot constitute a complete semantic system. If every word in our system is defined only in terms of other words, the system has no grounding. Some words also need to have sensory content of their own, i.e. a distinct identity, a referent. Even when modeling natural language processing at a higher level, without direct sensory input, it turns

Data	All	Synon.	$E_i < 0.15$	E_{avg}
Familiar	75	46	95	.038
Unfamiliar	75	41	94	.040
Complete	77	59	97	.023

Table 5: **Performance of basic FGREP.** The Familiar and Unfamiliar data sets are as before, Complete contains all sentences produced by the generators. The first column indicates the percentage of correct words out of all output words, the second shows the percentage for words with synonyms. The third column indicates the percentage of output units which were within 0.15 of the correct value, and the last column shows the average error per output unit.

out that an approximation of the sensory grounding is necessary to maintain distinct identities for the words.

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 any difference in the usage, the FGREP process will develop different representations for the words. 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 which 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 5), it produces an output pattern which 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. The results are shown in table 5.

Even though the system has learned the mapping task very well (97% of the output units within 0.15), it produces a pattern which is closest to the correct output word only about 77% of the time. Counting only the words which have synonyms (i.e. equivalent words such as **man**, **woman**, **boy**, **girl**), this is only 59%. 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 for enough differences to keep the representations separate. Even in the end the discrimination between synonymous words remains above chance level, because the training data was not completely symmetric, and minute differences remain in their representations. Also, the generalization networks of figure 12, which were trained with even more incomplete data, can separate the words much better.

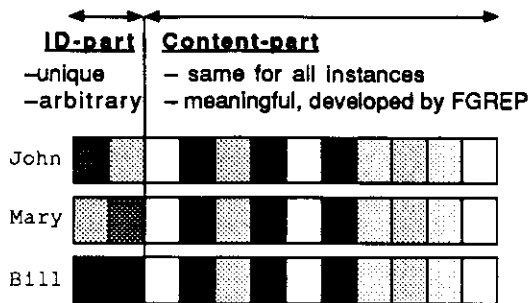


Figure 14: Cloning word instances. Instances John, Mary and Bill are created from the prototype word human.

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 which have similar meaning. It seems necessary to complement the meaning of the item, as extracted from its use, with an approximation of its sensory grounding.

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 now consists of two parts: the content part, which is developed in the FGREP process and which encodes the meaning of the word, and the ID part, which is unique for each instance of the same word (figure 14). The ID part has no intrinsic meaning in the system, but it distinguishes the word from all other similar words. The technique can be thought of as an approximation of sensory grounding, where the ID part stands for the sensory referent of the word.

Before training, a set of *prototype words* is selected, i.e. the words that we later want to make clones of (e.g. human). During training, the units within the ID part of the prototype words 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.

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) are then added to the lexicon, and they can be used for input/output. This technique makes it possible to deal with a large and open-ended set of semantically equivalent human names, foods, hitters, etc. without confusing them. This capability seems to be a prerequisite for modeling symbolic thought and logical reasoning (section 11.3).

6.3 Performance with cloned synonyms

The technique was tested by developing a prototype word for each of the word equivalence classes: human for {man, woman, boy, girl}, utensil for {fork, spoon}, predator for {wolf, lion}, fragile1 for {plate, window}, hitter1 for {ball, hatchet, hammer}, hitter2 for {paperwt,

Data	All	Synon.	$E_i < 0.15$	E_{avg}
Familiar	96	96	95	.043
Unfamiliar	97	98	94	.045
Complete	97	98	96	.038

Table 6: Performance of FGREP with cloned synonyms. The network was trained with 0.1 learning rate for 2000 epochs and with 0.05 until 5000, which took approximately the same time as training without cloning. The same test sets were used as in table 5. The average errors are higher partly because the ID components consisted of extreme values 0 and 1, which are harder to achieve than midrange values.

rock} and food1 for {cheese, pasta, carrot}. These words were replaced by their prototype in the training data and the identical sentences were removed. The new training set consists of 207 different sentences.

After training, each prototype was cloned to cover the original vocabulary, choosing binary values for the two ID units. The resulting set of representations is shown in figure 15. The representations reflect the categorization like before, and have the same processing characteristics. However, the members within each category are now more distinct.

Using the cloned representations the system is able to keep the equivalent words separate, as can be seen from table 6. The error in the output layer has not increased significantly, but 97% of all output words are now correct, with 98% of the words with synonyms.

With this particular task and data, the representations of words in neighboring categories, such as hitter1 and hitter2 become very similar. If the ID-patterns within the categories are very dissimilar, items are sometimes confused with items in close-by categories having the same ID-pattern. 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 technique can be applied to any word in the training data, and in principle, the number of clones per word is unlimited. This allows us to tremendously 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*. For example, the system can process sentences like Mary ate the chicken-soup with a silver-spoon and John downed the lasagna with a plastic-fork, which all have the same meaning: A human ate food with a utensil.

Figure 16 shows the performance data with an increasing number of clones for each word. The system was trained

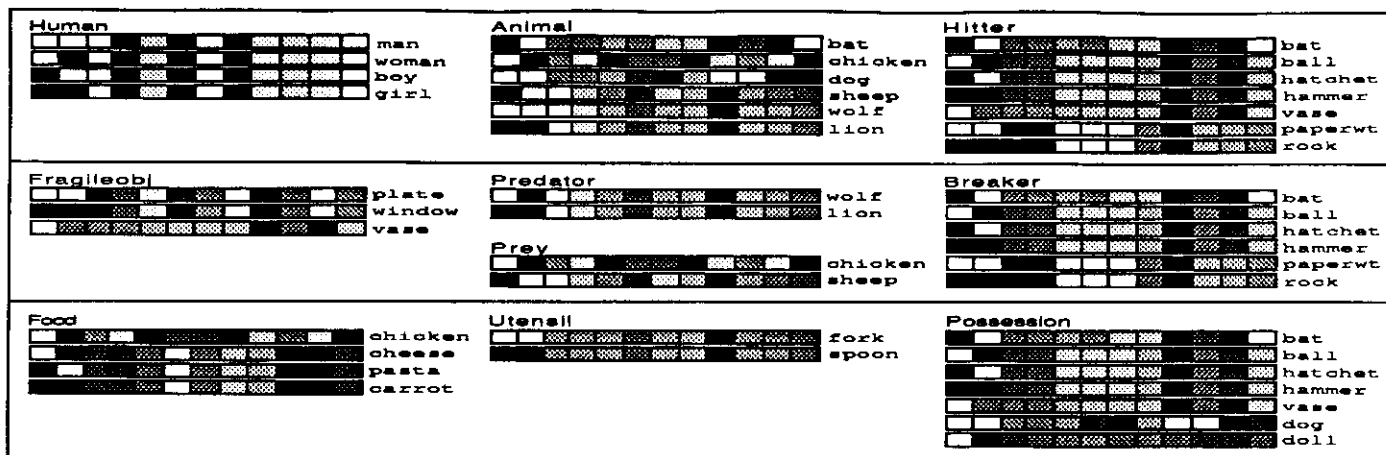


Figure 15: Final representations (cloned synonyms). The representations for the synonymous words {man, woman, boy, girl}, {fork, spoon}, {wolf, lion}, {plate, window}, {ball, hatchet, hammer}, {paperwt, rock} and {cheese, pasta, carrot} were formed by cloning a single prototype word. The first two units of the representation were used for the ID.

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 the figure, discrimination degrades approximately linearly as a function of clones. With sixteen different clones represented in 2-unit IDs the system still produces correct words 93% of the time. This is remarkable since the number of different sentences grows polynomially, proportional to the fourth power of the number of clones. For a single clone, there are 210 different sentences, four clones produce 27,024, sixteen 5,495,040 and thirty-six clones give us 134,401,680 different sentences.

Cloning word instances is very much like generating new symbols with the LISP 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 combinatorial number of inputs [Brousse and Smolensky, 1989] in a nontrivial task.

7 Processing sequential input/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

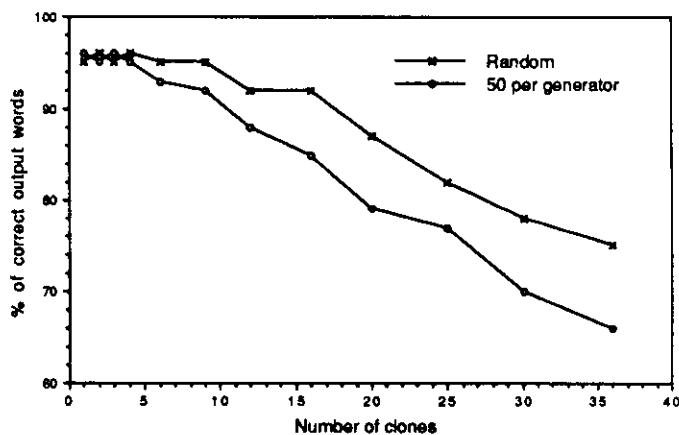


Figure 16: Word discrimination of basic FGREP with extended vocabulary. The horizontal axis shows the number of clones created for each word, and the vertical axis indicates the percentage of correct output words. The Random -data set consisted of 1000 sentences selected randomly among the set of all sentences, while the 50 per generator -data set consisted of an equal number of randomly chosen sentences from each generator. 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 0.035, with 97% of the output within 0.15, and for the 50 per generator -data set 0.040 and 95%.

is presented, allowing us to efficiently deal with sequential input data. The architecture is used to form case-role representations directly from word-by-word sequential natural language input without the need for preprocessing.

7.1 Encoding constituency in sequences

The system input/output can sometimes be made more efficient by representing complex data with higher-order structure as a sequence of constituents. In an assembly-based representation, 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 all sentences do not have these elements. Representation structure becomes a serious problem when we want to scale up and represent, e.g., 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 one of them. This leads to combinatorial explosion in the size of the representations.

On the other hand, assembly-based representation preserves the high-dimensional relations of the constituents, because assemblies are role-specific. The relation of a constituent to all other constituents is well-defined as soon as it is placed in a specific assembly. In many cases the relations are much simpler and all this representational power is not needed. For example, information about partial ordering of the elements might 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, I/O of complex data takes place exactly as in natural language, where complex thoughts are transformed into a sequence of words for transmission through the communication channel. Consequently, natural language input/output is most naturally and efficiently represented in this manner.

7.2 Recurrent FGREP module

In recurrent FGREP, the extension of FGREP to sequential input and output, the basic network architecture is similar to the one proposed in [Elman, 1988; Elman, 1989] (see also [Jordan, 1986; Servan-Schreiber *et al.*, 1989; St. John and McClelland, 1989]). 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 17). 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 out-

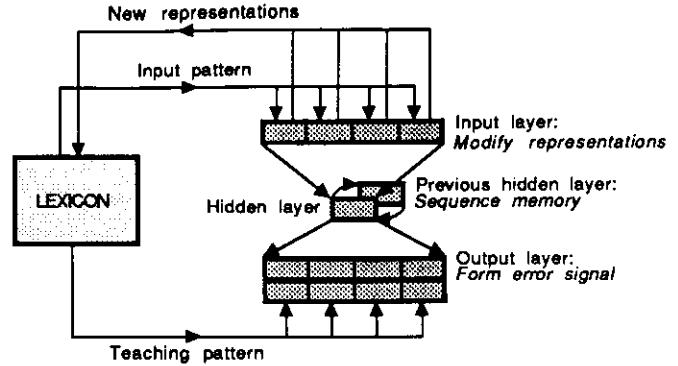


Figure 17: Recurrent FGREP-module. The hidden layer pattern is saved after each step in the sequence, and used as input to the hidden layer during the next step, together with the actual input.

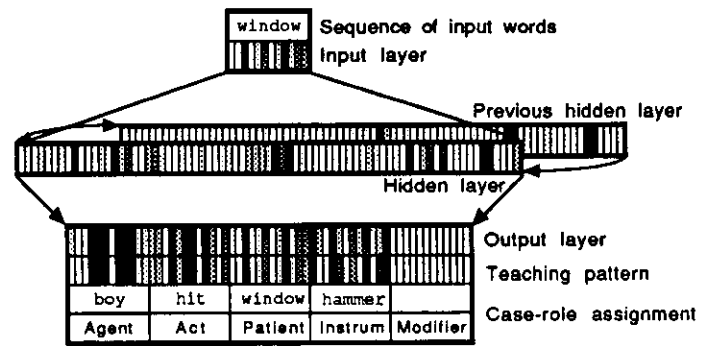


Figure 18: Snapshot of recurrent FGREP simulation. The system is in the middle of reading The boy hit the window with the hammer.

put representation, and for generating an output sequence from a stationary input. In a *sequential input network*, the actual input changes at each time step, while the teaching pattern stays the same. The network is forming a stationary representation of the sequence. In a *sequential output network*, the actual input is stationary, but the teaching pattern changes at each step. The network is producing a sequential interpretation of its input. The error is back-propagated and weights are changed at each step. Both kinds of recurrent FGREP networks are also developing 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 listed in figure 18. More resources are required to assign case-roles from sequential input than were from syntactic constituents. The hidden layer now serves also as a sequence memory, and more capacity is needed. The size of the hidden layer was increased to 75 units. Training times are longer, because the sequences must be learned also, and backpropagation is done after each item in the sequence.

Data	All	Synon.	$E_i < 0.15$	E_{avg}
Familiar	96	96	94	.042
Unfamiliar	97	97	94	.043
Complete	97	95	93	.045

Table 7: Average performance with cloned synonyms from sequential input. 0.1 learning rate was used during the first 1500 epochs, 0.05 until 2000 and 0.025 for another 100 epochs.

The input consists of the actual words of table 1, including words **the** and **with**. 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. The case-role representation of the sentence thus gradually forms at the output. Each sentence is ended with a period, which has its own representation in the lexicon. The case-role representation is complete after the period is input.

The performance figures are presented in table 7. Processing sequences does not significantly degrade the performance: the system now outputs 95% of the synonymous words correctly, with 93% of the output values within 0.15. 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 become clearly demonstrated. The expectations arise in the assemblies for unspecified case-roles (figure 18). These patterns are averages of all possible bindings at that point, weighted by how often they have occurred. When 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 18, 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 can have modifiers in the data (table 1). Most probably an instrument is going to be mentioned later, and a strong expectation of a **hitter** is displayed in the instrument slot. If **with** is read next, a hitter is certain to follow, and only the ID part will display an average pattern. Reading a period next instead would clear the expectation in the instrument slot, telling the system that the sentence is complete without this constituent.

The expectations emerge automatically and cumulatively from the input word representations. This can be used e.g. to fill in missing input information (section 9.8), or to guess the meaning of an unfamiliar word (section 10.3).

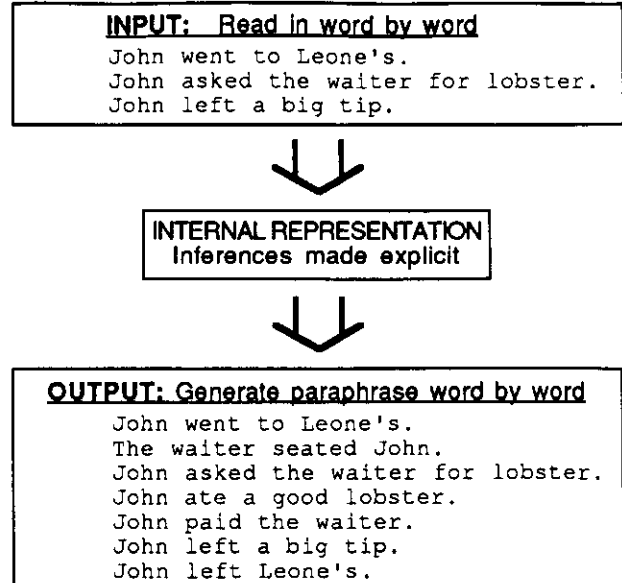


Figure 19: Generating a full paraphrase of an incomplete script-based story. This story is an instance of the fancy-restaurant track of the restaurant script. The input story is read word by word into the internal representation, where all the inferences are made explicit. The paraphrase is generated word by word from this internal representation.

8 A composite task: Paraphrasing script-based stories

The recurrent FGREP modules have been used as building blocks in more complex cognitive systems. An obvious task is the scale-up of sentence processing into the next higher level, e.g. 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 is given in figure 19.

The input story mentions only three events about John's visit to Leone's. 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.

Schank and Abelson suggested that this kind of knowledge about everyday routines could be organized in the form of scripts [Schank and Abelson, 1977]. Scripts are schemas of often encountered, stereotypical sequences of events. Common knowledge of this kind makes it possible to efficiently perform social tasks such as visiting a restaurant, visiting a doctor, traveling by airplane, shopping groceries, attending a meeting, etc. People have hundreds, maybe thousands of scripts at their disposal. Each script may divide further into established minor variations, or tracks. For example, there is a fancy-restaurant track,

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

Table 8: Representation of a script-based story as a causal chain and role bindings.

a fast-food track and a coffee-shop track for the restaurant script.

In symbolic artificial intelligence a script is represented as a causal chain of events with a number of open roles [Schank and Abelson, 1977; Cullingford, 1978; DeJong, 1979; Dyer *et al.*, 1987]. A script-based story is an instantiation of the script with specific role bindings (table 8). Applying knowledge about scripts to a story requires identifying the relevant script and filling in its roles with the actual words in the story. Once the script has been recognized and the roles have been instantiated, the sentences are matched against the events in the script. Events which are not mentioned in the story but are part of the causal chain can be inferred, as well as certain fillers of roles which were not specified in the story. For example, it is plausible to assume that John ate the lobster, the lobster tasted good, etc. The story can then be paraphrased in full from its representation.

The causal chain is a sum 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 representing stories. 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 represented as activity patterns in role-specific assemblies.

9 Connecting the building blocks in DISPAR

9.1 System architecture

The DISPAR system (DISTRIBUTED PARaphraser) [Miikula and Dyer, 1989b] consists of two parts (figures 20 and 21). The first part reads in 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/producing the word representations and the other for reading/producing 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 into the input layer of the next network. The input and output of each network consists of distributed representations of words, which are stored in a global lexicon.

9.2 Performance phase

Let us present the system with the first sentence of the example story, **John went to Leone's** (figure 20). The task of the sentence parser network is to form a stationary case-role representation of this sentence, as has been described in previous sections. 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, receiving a sequence of the sentence case-role representations as its input, and forming 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=Leone's`, `food=lobster`, `taste=good` and `tip=big`. This technique is making the best use of the fixed-size assembly-based representation. *The assemblies are not role-specific, but their interpretation varies with the data.*

The internal representation completely specifies the events of the script. The second part of the system is trained to paraphrase the story from the internal representation. The idea is simply to reverse the process of reading in.

The story generator network (figure 21) receives the complete slot-filler representation of the story as its input, and generates the case-role assignment of the first sentence of the story as its output. This output is fed into the sentence generator network, which produces the distributed representation of the first word of the first sentence as its output. Again, the hidden layer of the sentence generator network is copied into 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 produces the case-role representation of the second sentence. The process is repeated until the whole story has been output.

9.3 Training phase

A good advantage of the modular architecture can be made in training these networks (figure 22). The tasks of the

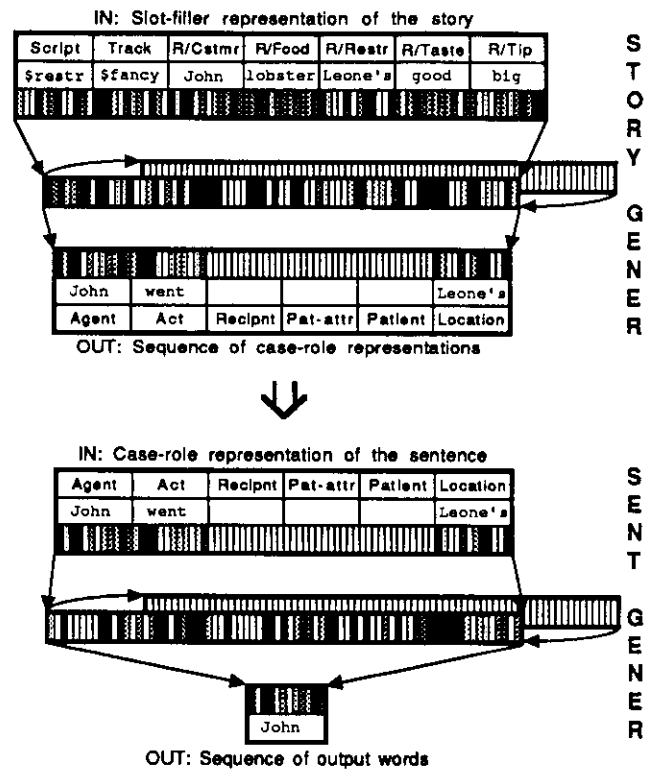
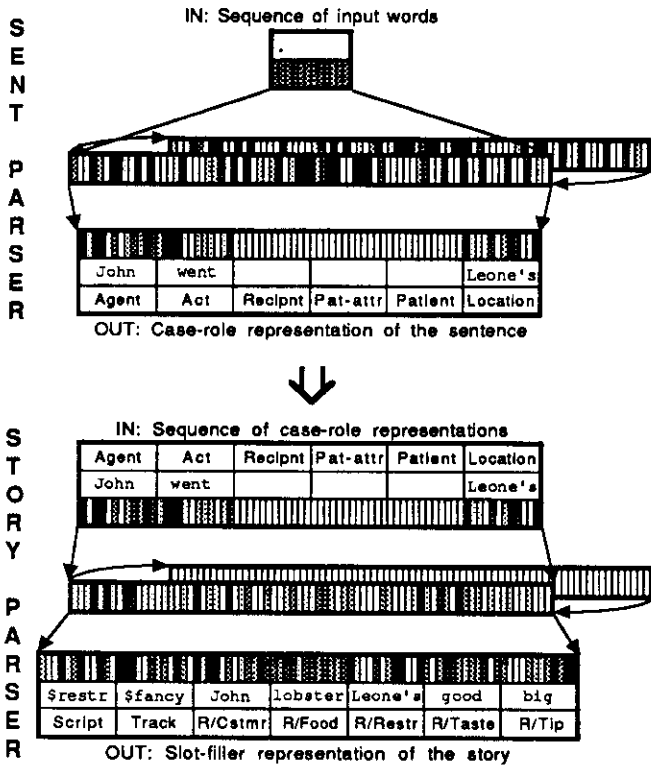


Figure 20: **Networks parsing the story.** The figure presents a snapshot of the simulation after the first sentence of the example story has been read. The script and the track have already been identified and a number of expectations about the role bindings are active at the output of the story parser. The role names (R/...) are specific for the restaurant script.

Figure 21: **Networks generating the paraphrase.** 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.

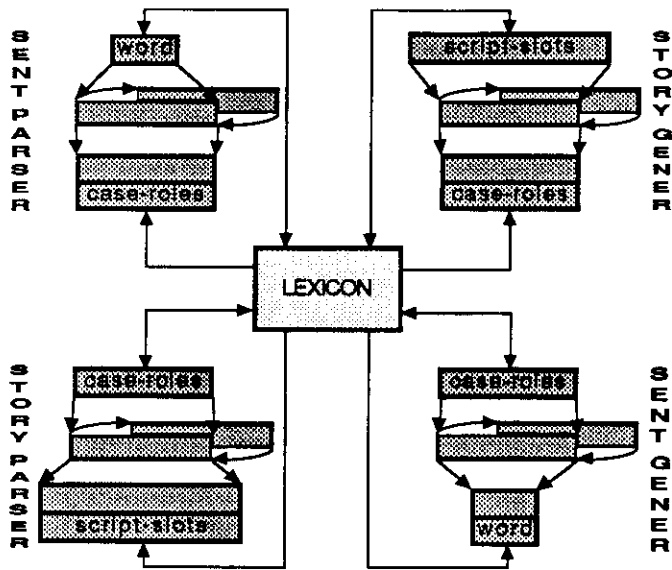


Figure 22: Training configuration. Each network is trained separately and simultaneously, developing the same lexicon.

four networks are separable, and they can be trained separately (e.g. on different machines) as long as compatible I/O material is used. The networks must be trained simultaneously, so that they are always using and developing the same representations.

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

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

9.4 Training data

The DISPAR system was trained to paraphrase stories based on restaurant, shopping and travel scripts. There are three tracks to each script, with four to five open roles. The story skeletons for each track are listed below, together with the role bindings.

Words in upper case are prototype words. The actual stories are formed from these skeletons by specifying the ID part for each prototype. In the training, the IDs for these words are assigned randomly for each story, but con-

sistently throughout the whole story. The test stories were generated by creating two instances from each prototype. The system was tested with two test sets: (1) complete stories, which included all sentences in the skeleton, and (2) incomplete stories, containing only the three sentences marked with "*".

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, \$train, PERSON,
TRAIN-ORIGIN, TRAIN-DEST, big/small

PERSON went to TRAIN-ORIGIN.*
PERSON bought a ticket to TRAIN-DEST.*
PERSON got-on the train.
The conductor punched the ticket.
PERSON traveled a big/small distance.
PERSON got-off at TRAIN-DEST.*

Bus-travel track

Fillers: \$travel, \$bus, PERSON, BUS-ORIGIN,
BUS-DEST, small

PERSON went to BUS-ORIGIN.*
PERSON waited for the bus.*
PERSON got-on the BUS-DEST bus.
PERSON paid the driver.
The bus arrived at BUS-DEST.*

PERSON got-off the bus.

The need for the script and track assemblies in the internal representation is obvious 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 script and track slots 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: good food \Leftrightarrow big tip, bad food \Leftrightarrow small tip. The system should be able to use these regularities to fill in unspecified roles.

9.5 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 into the next network. Instead, they were replaced by the correct patterns, obtained by concatenating the current word representations in the lexicon. This way each network was trained the same number of epochs. (For an alternative training method, where the networks are actually 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. Most of the training effort was expended on preparing the system for the different IDs. Without IDs a satisfactory error level would have been achieved in about 2,000 epochs.

9.6 Representations

There are very few similarities in the resulting representations. The vocabulary in the example stories is quite large and consists mainly of words which have a very specific use. As a result, their 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 in the end. On the other hand, in the early stages of training, sentences are often ended prematurely with a period, and after that, only periods are output. This hap-

Network	All	Synon.	$E_i < 0.15$	E_{avg}
Sentence parser	99	99	99	.017
Story parser	99	99	96	.023
Story gener	99	99	97	.021
Sentence gener	99	99	96	.037

Table 9: Average performance of DISPAR networks with complete input stories. Two instances for each prototype word were cloned, generating a total of 96 different test stories.

pens because the representation for the period is modified at the end of every sentence, where reading it in should have very little effect on the output. Its representation becomes the most neutral representation, i.e. it contains many components close to 0.5 (see e.g. figure 20). 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, extracted from the input examples. It seems that this approach could be used to deal with 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.

9.7 Role binding

In the performance phase two instances were created from each prototype word. The set of test stories was put together from all combinations of the instances, 96 stories altogether (including the two different versions of tip and distance for the fancy-food, coffee-shop-food and train-travel tracks). The performance of the system was tested with these completely specified stories. Table 9 presents performance figures for each network. Even when the networks are connected in a chain (output of one network feeding the input of the next), the errors do not cumulate in the chain. The noise in the input is efficiently filtered out, and each network performs approximately at the same level.

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 etc. 99% of the time. Especially interesting is, that once a role binding (e.g. customer=John) is selected in an earlier event, even if it is incorrect it is usually maintained throughout the paraphrase generation. Thus, DISPAR performs plausible role bindings - an essential task in high-level inferencing and postulated as very difficult for PDP systems to achieve [Dyer, 1989].

9.8 Paraphrasing incomplete stories

The training corpus consisted of complete stories, and the system was trained to reproduce a story exactly as it was input. An interesting question is how well the system can fill in the events of a story which consists of only a few

Network	All	Synon.	$E_i < 0.15$	E_{avg}
Sentence parser	99	99	98	.019
Story parser	96	97	93	.041
Story gener	99	97	97	.027
Sentence gener	99	97	95	.044

Table 10: Average performance of DISPAR networks with incomplete input stories. The same set of stories were used as in table 9, but only the sentences marked with "*" in the story skeletons were included.

sentences.

The system performs very well in this respect (table 10). There is very little degradation in performance compared to the complete stories. Stories of "natural" length, like our example, 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 between the possible choices. This follows directly from the tendency to generate expectations.

For example, if the story consists of only the sentence John went to Leone's, the food quality can be inferred (good, because Leone's is a fancy-restaurant), but an intermediate representation develops in the food-slot (figure 20). In paraphrasing the story it seems as if one of the possible fancy-foods is chosen at random. This choice is usually consistent throughout the story, because all sentences are generated from the same pattern in the food-slot.

In general it seems that a network which builds a stationary representation of a sequence might be quite sensitive to omissions and changes in the sequence. Each input is interpreted against the current position in the sequence by combining it with the previous hidden layer. If items are omitted from the sequence it seems that the system could very easily lose track. But this is not the case in the script reader. The network was trained to always shoot for the complete output representation, and the hidden layer pattern stabilizes very early in the sequence. Reading subsequent input items has 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 the earlier sections) 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 in the sequence which 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.

Once the script has been instantiated and the role bindings fixed there is no way of knowing which of the 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 which were not mentioned are often answered as if they were part of the original story.

9.9 Extensions to script processing architecture

Pronoun reference is not a particularly hard problem in understanding script-based stories. People do not get confused when reading e.g.: **The waiter seated John. He asked him for lobster. He ate the lobster.** 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 representation for these words will develop into a general actor, food etc.

A mechanism should be developed 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 and Smolensky, 1989]. Instantiation of a script would now have the form of a tensor product "cube" [Dolan, 1989]. One face of the cube would stand for the script and track, one for the role, and one for the filler. It would be possible to represent multiple simultaneously active scripts in the same cube. Scripts could be partially activated and their boundaries would be less rigid.

It might be possible to model learning new scripts and tracks with a self-organizing system, where often encountered sequences of events gradually become rigid stereotypical memories. It seems that hierarchical feature maps could be used for this task [Miikkulainen, 1989]. Combining the feature map mechanism with the current architecture is a very interesting research direction.

10 Towards more advanced models

10.1 Making use of modularity

Most of the parallel distributed processing (PDP) models have relied on capabilities of homogeneous architectures, such as simple backpropagation networks. It seems that in more complex domains, division into subtasks could be a useful approach [Minsky, 1985; Ballard, 1987; Waibel *et al.*, 1988].

For example, it might be possible to treat the script paraphrasing task as a pattern completion problem, and use a single, flat backpropagation network without teaching at the intermediate levels (see e.g. [Harris and Elman, 1989]). Building the system from hierarchically organized separable modules provides two major advantages: (1) the

System	Time	Words	$E_i < 0.15$	E_{avg}
DISPAR parsers	1.5	96	98	.022
Single Rec. FGREP	3.5	91	98	.021
Single Rec. FGREP	7.5	93	99	.013

Table 11: **Reducing complexity with hierarchical modules.** The training data consisted of restaurant script based stories, without IDs. Training time is shown in days on an HP 9000/350 workstation. Learning rate was 0.1 at first, and 0.05 after 3.5 days.

task is effectively divided into subgoals, which is an efficient way to reduce complexity [Korf, 1987], and (2) the system is forced to develop meaningful internal representations, which can be used by other systems, e.g. a question answering network.

A simple comparison of hierarchical networks 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 internal slot filler representation. The hierarchical system achieved a satisfactory level of error about twice as fast as the simple system (table 11). The flat 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 scriptal knowledge which is needed for inferencing, while the sentence networks form the specific language interface. 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 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*.

Question answering could be implemented as a separate module which receives as its input the slot-filler representation of the story, together with the representation of the question which has been read in sequentially by the sentence parser network. The question answering module generates a case-role representation of the answer sentence, which is then output word by word with the sentence generator network. Using the previous hidden layer of the

System	Time	Words	$E_i < 0.15$	E_{avg}
DISPAR parsers	1.5	96	98	.022
Without FGREP	1.5	96	99	.022

Table 12: **Effect of FGREP on training time.** The "Without FGREP" -system uses the final representations of the DISPAR parsers, without modifying them. The same training data was used as in table 11.

System	Time	Words	$E_i < 0.15$	E_{avg}
DISPAR parsers	1.0	96	97	.025
Full DISPAR	0.7	96	99	.014

Table 13: **Effect of more modules on training time.** The same training data was used as in table 11, but each network was trained on separate workstations.

answer-producing network as the context to the question, context effects on question answering [Lehnert, 1978] could be modeled. Propagating the question to the slot-filler representation could have an effect on the ambiguous slots, i.e. the questions could modify the memory [Loftus, 1975].

10.2 The role of the central lexicon

Developing the input/output representations in a central lexicon does not seem to incur an extra cost on the training time. The lexicon of the hierarchical script reader system (section 10.1) 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 (table 12).

Adding more modules to the system, all developing the same lexicon, does not seem to make learning any harder either. The parser part of the paraphraser system (4 modules on 4 workstations) reached a satisfactory level of performance 30% faster than the mere script reader system (2 modules on 2 workstations) (table 13). Apparently, in a system with more modules, the representations become descriptive faster, speeding up the total learning.

In the experiments discussed in this article, the lexicon consists of 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 into 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 event order and specific roles), although they are processed like words by the system.

The FGREP representations encode general semantic knowledge, extracted as regularities over a number of examples. In a more complex model, other kinds of information need to be stored also. For example, we might want to save representations of specific stories in an episodic memory. These memories cannot be addressed directly with a unique index like words in the lexicon. Each episode (i.e. story) is different and a unique experience, and the only handle to the episode is its content. The story memory must be content-addressable, an associative memory -type architecture. Possible implementations include Hopfield networks [Hopfield, 1982], brain state in a box (BSB) -model [Anderson and Mozer, 1981], Boltzmann machine [Ackley *et al.*, 1985] and sparse distributed memory [Kanterva, 1988]. Some modules in the system, like the question answerer, could use both lexicon representations and the representations from the associative memory as their input/output. The episodic memory would exhibit interference effects from similar stories, spurious memories etc. This is desirable, as long as the memory errors are plausible in terms of human performance.

10.3 Extending the lexicon

The system could extend its lexicon dynamically when needed. For example, the word **bat** has two meanings in our data, yet the system is trying to code both in a single representation. There should be a way to recognize a situation like this and replace **bat** with two entries, **live-bat** and **bb-bat**. When the word **bat** is encountered in the text, a choice must be made on which of the representations to access. The expectations generated by the network could be used to outline the representation, and the one closest to it then fetched from the lexicon.

The same mechanism could be used to acquire new words. Each time a new word is encountered, a new entry would be created in the lexicon. Expectations generated by the network could be used to come up with an initial guess for the representation. The expectation pattern in the appropriate slot (specified by the teaching 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 **Leone's** (as a new word) would probably be very much like **fancy-restaurant**, the initial lexicon entry for **Leone's** would be an instance of **fancy-restaurant**.

The initial entry for the new word would be modified through experience, and it would acquire content of its own. The instance would gradually become a prototype

itself. Eventually there would be several different fancy restaurant names in the vocabulary, each with an established unique meaning. The original fancy-restaurant representation would still be in the lexicon, standing for a generalized form of these specific instances. This mechanism of learning new words could thus model forming of subclass hierarchies, i.e. more specific terms from general terms.

10.4 Cumulative representations

The lexicon representations adapt to the task and data. If the training data is changed, the representations will adapt, and reflect the new requirements. What was previously learned will gradually fade away. In some applications this is appropriate, but in others (e.g. in learning the meaning of words) the effect should be mostly cumulative. The old information should become gradually more rigid.

One possible way to achieve this is to use weights with different learning rates [Hinton and Plaut, 1987]. Another possibility would be to let the size of the representation grow as more data comes in. The fastest learning would occur in the new area, while older areas changed slower. New information would be learned in terms of what is already known. If the input is familiar, the representation would be changed within the current area and would interfere with the previously learned. If the input is very different from the previous ones, new units would be acquired to prevent unlearning previous information. Periodically the representation could be reorganized to use the units more efficiently.

10.5 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.

The sensory form of the word (i.e. letters or sound) is a unique index to its lexicon representation. A network could be trained to obtain the lexicon entry for each input word. If the lexicon is implemented as a list of word assemblies, the task reduces to activating the correct assembly. Simple routing networks could copy this pattern to the appropriate input and teaching assemblies, and transport the modified representations back into the lexicon. At the output, a winner-take-all network would select the lexicon entry which is nearest to the output pattern. Attached to the lexicon entry there would be a network which outputs the corresponding sensory form of that word.

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 [Rumelhart *et al.*, 1986a; Pollack, 1987].

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, an-

other 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.

11 Discussion

11.1 PDP vs. symbolic AI

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 same architecture can learn to process a wide variety of inputs, depending on the training data. A network which was trained to paraphrase restaurant scripts can learn to paraphrase travel stories, shopping stories etc. just as well.

This contrasts with the traditional symbolic artificial intelligence models, where the processing instructions must be hand-crafted 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 into previously unseen inputs is possible only if there exists a specific rule that specifies how this is done. Representations of these rules and their applicability is often very complex. This 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.*

11.2 Connections to biological systems

The modular FGREP architecture was not developed with direct biological plausibility as a goal. Whether there exists a separate lexical memory in neural tissue, as a list of word representations or as an associative memory, remains an open question, although there are some indications in this direction. Recent recording results [Heit *et al.*, 1989] suggest that neurons in the human hippocampus respond to visually presented individual words similarly to the representation units (figure 8).

What neural mechanisms could be responsible for modifying the representations, and how processing with them could be learned, is a completely open question. It is far from clear what neural mechanisms backpropagation itself could correspond to. Some connections can be made at higher level: the error signal could represent some unspec-

ified form of feedback, teaching patterns could be formed from related sensory and memory inputs, the lexicon and modules could correspond to processing centers, etc.

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 and is aimed at explaining how higher-level behaviour can emerge as a result of such processes. Another level of models is needed, explaining the current models in more plausible terms, before any direct correspondence to neural mechanisms can be claimed.

11.3 Symbolic processing in PDP

The mechanism of sensory-level IDs is a first step towards grounding symbolic reasoning in parallel distributed processing. Any symbolic system needs to be able to deal with two kinds of information: (1) semantics of symbols, i.e. knowledge about the properties of symbols and relationships between them, and (2) identities of symbols, based on some arbitrary surface-level tag, e.g. sensory perception of the referent (see also [Harnad, 1989]). 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), 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 instances inherit the properties of 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 sensory-ID approach the identity and semantic content are *kept together in a single representation* and processed through the same pathways and structures. 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. Processing the IDs is hard because the rest of the input pattern provides no cue about what the ID should be. To produce a correct ID pattern in the output, 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 learn-

ing 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 to specific instances only 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 two 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 boundary between ID and content is less well defined.

11.4 Parallel distributed inference

Inference is often modeled in artificial intelligence systems based on probabilistic formalisms ([Pearl, 1988; Duda *et al.*, 1977; Buchanan and Shortliffe, 1985]). Events are known with certain probabilities and they provide evidence for other events. 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. *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 find the most relevant data from an enormous collection of information, and build a hypothesis based on that, knowing that this 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, an algorithm. 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, i.e. 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, 1989]). The network extracts statistical knowledge from the input data, and these statistics are brought together to generate the inference. 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 which is the average of all possible answers, weighted by their probabilities.

12 Summary

An architecture for connectionist natural language processing is presented which consists of hierarchically organized independent subnetworks and a central lexicon. The I/O of the subnetworks consists of distributed representations stored in the lexicon, and the modules communicate using these representations.

The representations are developed automatically by the FGREP-mechanism while the network is learning the processing task. With backward error propagation extended to the input layer, the representations are developed automatically as if they were an extra layer of weights. FGREP creates a reactive training environment, i.e. the required input/output mappings change as the I/O representations change.

There are no identifiable microfeatures nor discrete categories in the resulting representations. All aspects of an input item are distributed over the whole set of units in a holographic fashion, making the system particularly robust against damage. Each representation also carries expectations about its possible contexts. The representations evolve to improve the system's performance in the processing task and therefore efficiently code the underlying relations relevant to the task. This results in extremely good generalization capabilities.

The lexicon can be extended by cloning new instances of the items, i.e. generating a number of items with the same properties but with distinct identities. This is accomplished by combining the semantic representation with a unique ID-representation. The technique is motivated by sensory grounding of words, and forms a basis for symbolic processing in PDP. It is possible to approximate a large number of items by dividing them into equivalence classes, resulting in combinatorial processing capabilities with linear cost.

Representing input/output as sequences overcomes the combinatorial explosion in representing structurally complex data. The technique is implemented in a recurrent FGREP network. This module, together with a central lexicon, can be used as a 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. The system is able to assign case roles correctly, indicate degree of confidence when the sentence is ambiguous, and generalize correctly for unfamiliar sentences. A hierarchical organization of four modules was trained to paraphrase script-based stories, again with natural language input and output. The complexity of this task is reduced by effectively dividing it into subgoals. Each module can be trained separately and in parallel, each developing the same lexicon. The system, DISPAR, is able to produce a fully expanded paraphrase of the story

from only a few sentences, i.e. the unmentioned events are inferred. The word instances are correctly bound to their roles, and simple plausible inferences of the variable content of the story are made in the process.

The main advantage of the approach over symbolic artificial intelligence systems is that the processing knowledge is extracted automatically from examples. The same architecture can learn to process a wide variety of inputs and make advantage of the implicit statistical regularities in the data, without having to be specifically programmed with the particular data in mind.

References

- [Ackley *et al.*, 1985] David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science*, (9):147-169, 1985.
- [Anderson and Mozer, 1981] James A. Anderson and Michael C. Mozer. Categorization and selective neurons. In *Parallel Models of Associative Memory*, Lawrence Erlbaum Associates, 1981.
- [Ballard, 1987] Dana H. Ballard. Modular learning in neural networks. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1987.
- [Bower *et al.*, 1979] Gordon H. Bower, John B. Black, and Terrence J. Turner. Scripts in memory for text. *Cognitive Psychology*, (11):177-220, 1979.
- [Brousse and Smolensky, 1989] Olivier Brousse and Paul Smolensky. Virtual memories and massive generalization in connectionist combinatorial learning. In *Proceedings of the Eleventh Annual Cognitive Science Society Conference*, Cognitive Science Society, Lawrence Erlbaum Associates, Hillsdale, NJ, 1989.
- [Buchanan and Shortliffe, 1985] Bruce G. Buchanan and Edward H. Shortliffe, editors. *Rule Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Reading, MA, 1985.
- [Cullingford, 1978] R. E. Cullingford. *Script Application: Computer Understanding of Newspaper Stories*. PhD thesis, Department of Computer Science, Yale University, 1978. Technical Report 116.
- [DeJong, 1979] G. F. DeJong. *Skimming Stories in Real Time: An Experiment in Integrated Understanding*. PhD thesis, Department of Computer Science, Yale University, 1979. Research Report 158.
- [Dolan, 1989] Charles Patrick Dolan. *Tensor Manipulation Networks: Connectionist and Symbolic Approaches to Comprehension, Learning and Planning*. PhD thesis, Computer Science Department, UCLA, 1989.
- [Dolan and Smolensky, 1989] Charles P. Dolan and Paul Smolensky. Implementing a connectionist production system using tensor products. In David S. Touretzky, Geoffrey E. Hinton, and Terrence J. Sejnowski,

- editors, *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1989.
- [Duda *et al.*, 1977] R. O. Duda, P. E. Hart, and N. J. Nilsson. *Development of a Computer-Based Consultant for Mineral Exploration*. Technical Report, Stanford Research Institute, 1977.
- [Dyer, 1983] Michael G. Dyer. *In-Depth Understanding: A Computer Model of Integrated Processing for Narrative Comprehension*. MIT Press, Cambridge, MA, 1983.
- [Dyer, 1989] Michael G. Dyer. Symbolic NeuroEngineering for natural language processing: A multilevel research approach. In J. Barnden and Jordan Pollack, editors, *Advances in Connectionist and Neural Computation Theory*, Ablex Publ., 1989. (in press).
- [Dyer *et al.*, 1987] Michael G. Dyer, R. E. Cullingford, and Sergio Alvarado. Scripts. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, John Wiley and Sons, New York, 1987.
- [Elman, 1988] Jeffrey L. Elman. *Finding Structure in Time*. Technical Report 8801, Center for Research in Language, University of California, San Diego, 1988.
- [Elman, 1989] Jeffrey L. Elman. Structured representations and connectionist models. In *Proceedings of the Eleventh Annual Cognitive Science Society Conference*, Cognitive Science Society, Lawrence Erlbaum Associates, Hillsdale, NJ, 1989.
- [Fillmore, 1968] Charles J. Fillmore. The case for case. In Emmon Bach and Robert T. Harms, editors, *Universals in Linguistic Theory*, Holt, Rinehart and Winston, New York, 1968.
- [Harnad, 1989] Stevan Harnad. The symbol grounding problem. In *CNLS Conference on Emergent Computation*, 1989.
- [Harris and Elman, 1989] Catherine L. Harris and Jeffrey L. Elman. Representing variable information with simple recurrent networks. In *Proceedings of the Eleventh Annual Cognitive Science Society Conference*, Cognitive Science Society, Lawrence Erlbaum Associates, Hillsdale, NJ, 1989.
- [Hartigan, 1975] John A. Hartigan. *Clustering Algorithms*. Wiley, 1975.
- [Heit *et al.*, 1989] Gary Heit, Michael E. Smith, and Eric Halgren. Neural encoding of individual words and faces by the human hippocampus and amygdala. *Nature*, (333):773-775, 1989.
- [Hinton, 1981] Geoffrey E. Hinton. Implementing semantic networks in parallel hardware. In Geoffrey E. Hinton and James A. Anderson, editors, *Parallel Models for Associative Memory*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1981.
- [Hinton, 1986] Geoffrey E. Hinton. Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Cognitive Science Society Conference*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.
- [Hinton and Plaut, 1987] Geoffrey E. Hinton and David C. Plaut. Using fast weights to deblur old memories. In *Proceedings of the Ninth Annual Cognitive Science Society Conference*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.
- [Hinton *et al.*, 1986] Geoffrey E. Hinton, James L. McClelland, and David E. Rumelhart. Distributed representations. In David E. Rumelhart and James L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume I: Foundations*, MIT Press, Cambridge, MA, 1986.
- [Hopfield, 1982] John J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences, USA*, pages 2554-2558, 1982.
- [Inhelder and Piaget, 1958] Brbel Inhelder and Jean Piaget. *The Growth of Logical Thinking from Childhood to Adolescence*. Basic Books, New York, 1958.
- [Jordan, 1986] Michael I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Cognitive Science Society Conference*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.
- [Kanerva, 1988] Pentti Kanerva. *Sparse Distributed Memory*. MIT Press, Cambridge, MA, 1988.
- [Kohonen, 1984] Teuvo Kohonen. *Self-Organization and Associative Memory*, chapter 5. Springer-Verlag, Berlin; New York, 1984.
- [Korf, 1987] Richard E. Korf. Planning as search: A quantitative approach. *Artificial Intelligence?*, 33(1):65-88, 1987.
- [Lehnert, 1978] Wendy G. Lehnert. *The Process of Question Answering*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1978.
- [Loftus, 1975] E. F. Loftus. Leading questions and the eyewitness report. *Cognitive Psychology*, (7), 1975.
- [McClelland and Kawamoto, 1986] James L. McClelland and Alan H. Kawamoto. Mechanisms of sentence processing: Assigning roles to constituents. In James L. McClelland and David E. Rumelhart, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume II: Psychological and Biological Models*, MIT Press, Cambridge, MA, 1986.
- [Miikkulainen, 1989] Risto Miikkulainen. *Script recognition with Hierarchical Feature Maps*. Technical Report UCLA-AI-89-10, Artificial Intelligence Laboratory, Computer Science Department, University of California, Los Angeles, 1989.
- [Miikkulainen and Dyer, 1988] Risto Miikkulainen and Michael G. Dyer. Forming global representations with extended backpropagation. In *Proceedings of the IEEE Second Annual International Conference on Neural Networks*, IEEE, 1988.
- [Miikkulainen and Dyer, 1989a] Risto Miikkulainen and Michael G. Dyer. Encoding input/output representations in connectionist cognitive systems. In David S.

- Touretzky, Geoffrey E. Hinton, and Terrence J. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1989.
- [Miikkulainen and Dyer, 1989b] Risto Miikkulainen and Michael G. Dyer. A modular neural network architecture for sequential paraphrasing of script-based stories. In *Proceedings of the International Joint Conference on Neural Networks*, IEEE, 1989.
- [Minsky, 1985] Marvin Minsky. *Society of Mind*. Simon and Schuster, New York, 1985.
- [Osherson, 1974] Daniel N. Osherson. *Logical Abilities in Children*. Lawrence Erlbaum Associates, 1974.
- [Pearl, 1988] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1988.
- [Pollack, 1987] Jordan B. Pollack. Cascaded back-propagation on dynamic connectionist networks. In *Proceedings of the Ninth Annual Cognitive Science Society Conference*, Cognitive Science Society, Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.
- [Pollack, 1988] Jordan B. Pollack. *Recursive Auto-Associative Memory: Devising Compositional Distributed Representations*. Technical Report MCCS-88-124, Computing Research Laboratory, New Mexico State University, 1988.
- [Quillian, 1967] M. Ross Quillian. Word concepts: A theory and simulation of some basic semantic capabilities. *Behavioral Science*, (12):410-430, 1967.
- [Rumelhart and McClelland, 1987] David E. Rumelhart and James L. McClelland. Learning the past tenses of English verbs: Implicit rules or parallel distributed processing. In B. MacWhinney, editor, *Mechanisms of Language Acquisition*, Erlbaum, Hillsdale, NJ, 1987.
- [Rumelhart et al., 1986a] David E. Rumelhart, Geoffrey E. Hinton, and James L. McClelland. A general framework for parallel distributed processing. In David E. Rumelhart and James L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume I: Foundations*, MIT Press, Cambridge, MA, 1986.
- [Rumelhart et al., 1986b] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In David E. Rumelhart and James L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume I: Foundations*, MIT Press, Cambridge, MA, 1986.
- [Schank and Abelson, 1977] Roger Schank and Robert Abelson. *Scripts, Plans, Goals, and Understanding - An Inquiry into Human Knowledge Structures. The Artificial Intelligence Series*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1977.
- [Schank and Riesbeck, 1981] Roger Schank and Christopher K. Riesbeck, editors. *Inside Computer Understanding*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1981.
- [Sejnowski and Rosenberg, 1987] Terrence J. Sejnowski and Charles R. Rosenberg. Parallel networks that learn to pronounce English text. *Complex Systems*, (1):145-168, 1987.
- [Servan-Schreiber et al., 1989] David Servan-Schreiber, Axel Cleeremans, and James L. McClelland. Learning sequential structure in simple recurrent networks. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1989.
- [St. John and McClelland, 1989] Mark F. St. John and James L. McClelland. Applying contextual constraints in sentence comprehension. In David S. Touretzky, Geoffrey E. Hinton, and Terrence J. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1989.
- [Sumida and Dyer, 1989] Ronald A. Sumida and Michael G. Dyer. Storing and generalizing multiple instances while maintaining knowledge-level parallelism. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1989.
- [Touretzky, 1989] David S. Touretzky. *Connectionism and Compositional Semantics*. Technical Report CMU-CS-89-147, Computer Science Department, Carnegie-Mellon University, 1989.
- [Waibel et al., 1988] Alex Waibel, Hidefumi Sawai, and Kiyohiro Shikano. *Modularity and Scaling in Large Phonemic Neural Networks*. Technical Report TR-I-0034, Advanced Telecommunication Research Institute, International Interpreting Telephony Research Laboratories, 1988.