Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

A MODULAR NEURAL NETWORK ARCHITECTURE FOR SEQUENTIAL PARAPHRASING OF SCRIPT-BASED STORIES

Risto Miikkulainen
Michael G. Dyer

January 1989
CSD-890010

A Modular Neural Network Architecture for Sequential Paraphrasing of Script-Based Stories

Risto Miikkulainen
Michael G. Dyer

February 1989

Technical Report UCLA-AI-89-02

A MODULAR NEURAL NETWORK ARCHITECTURE FOR SEQUENTIAL PARAPHRASING OF SCRIPT-BASED STORIES *

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

We have applied sequential recurrent neural networks to a fairly high-level cognitive task, i.e. paraphrasing script-based stories. Using hierarchically organized modular subnetworks, which are trained separately and in parallel, the complexity of the task is reduced by effectively dividing it into subgoals. The system uses sequential natural language input and output, and develops its own I/O representations for the words. The representations are stored in an external global lexicon, and they are adjusted in the course of training by all four subnetworks simultaneously, according to the FGREP-method. By concatenating a unique identification with the resulting representation, an arbitrary number of instances of the same word type can be created and used in the stories. The system 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.

1 Introduction

Consider the following short story:

John went to Leone's. John asked the waiter for lobster. John gave a large tip.

Based on our experience, we can fill in a number of events and paraphrase the story in more detail:

John went to Leone's. The waiter seated John. John asked the waiter for lobster. The waiter brought John the lobster. John ate the lobster. The lobster tasted good. John paid the waiter. John gave a large tip. John left Leone's.

In doing this, we have used our general experience of restaurants and the stereotypical events that occur in a visit to a restaurant. Schank and Abelson proposed that this knowledge of everyday routines is 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 a visit to a restaurant, a visit to a doctor, traveling by airplane, attending a meeting, etc. People have hundreds, maybe thousands of scripts at their disposal. Each script may divide further into different variations, or tracks. For example, there is a fancy-restaurant track, a fast-food track and a coffee-shop track for the restaurant script.

In machine understanding of stories based on scriptal knowledge the script is represented as a causal chain of events with a number of open roles [Schank and Abelson, 1977], [Cullingford, 1978]. Applying this knowledge to a situation requires identifying the relevant script and matching the roles with the situation. Entering, seating, ordering, getting food, eating, paying, tipping and leaving form a causal chain for the restaurant script. The roles in this script are customer, restaurant, food, quality of food, size of tip, etc. These roles are filled with John, **Leone's**, **lobster**, etc. in the story above. 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. For example it is plausible to assume that John ate the lobster, the lobster tasted good, etc.

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 unit activities.

This paper describes a system which was trained to paraphrase script-based restaurant stories. Our main goal was to show that neural networks have the right properties for representing and applying script-like knowledge. *Inferring unmentioned events, inferring the most likely role fillers, and disambiguating pronoun references emerge naturally in the network approach.* We also wanted to demonstrate that the network can be trained with natural language input and output and without hand-coded microfeature information. The network "reads" the story sequentially, word by word, and produces the paraphrase word by word. *The distributed representations of the words are developed by the network itself while it is learning the paraphrasing task.* The third goal was to devise a scheme for binding

*This research was supported in part by an ITA Foundation grant, and by the JTF program of the DoD, monitored by JPL. The first author was also supported in part by grants from the Academy of Finland, Finnish Cultural Foundation and the Finnish Science Academy. The simulations were carried out on equipment donated to UCLA by Hewlett Packard. Special thanks go to Trent Lange for valuable comments on an earlier draft of this paper.

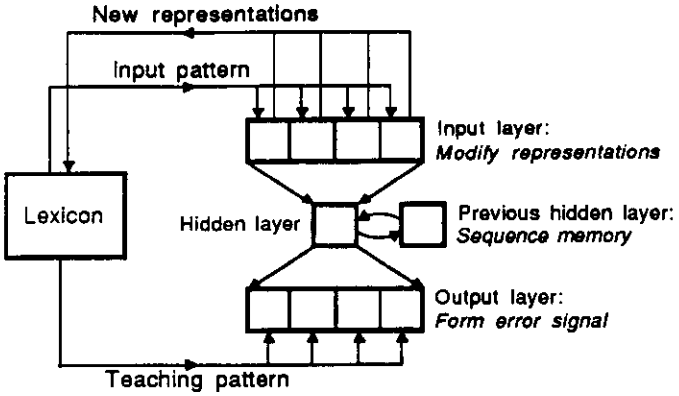

Figure 1: Sequential FGREP-module.

a large number of different fillers to their corresponding roles. By concatenating a unique identification with the prototype representation developed during training, an arbitrary number of instances of the same word type can be created. Lastly, it was necessary to divide the task into meaningful, simpler subtasks and develop a modular architecture which could be trained more efficiently.

2 System architecture

2.1 Sequential recurrent FGREP - a building block

The sequential recurrent extension of the FGREP-mechanism (Forming Global Representations with Extended backPropagation, [Miikkulainen and Dyer, 1988b]) presented here is based on a basic three-layer backward error propagation network (figure 1). The network learns the processing task by adapting the connection weights according to the standard backpropagation equations [Rumelhart *et al.*, 1986, 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 concepts are represented and modified simultaneously.

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

The *process begins with a randomly generated lexicon* containing no pre-encoded information. *During the course of learning, the lexical representations adapt to reflect the implicit regularities of the task.* Single units in a resulting representation for a word do not necessarily have a clear interpretation, but the 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.

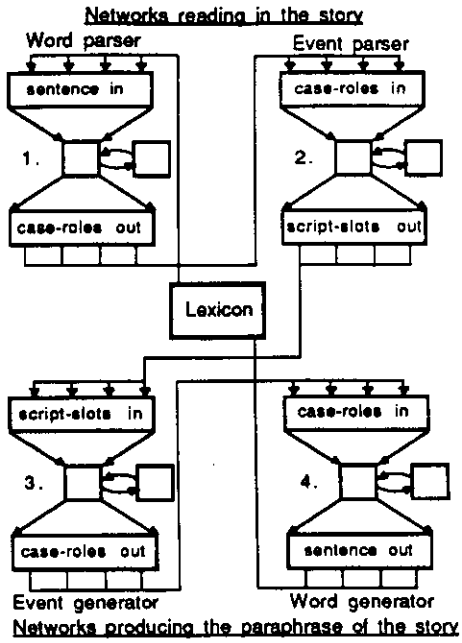

Figure 2: Performance configuration.

Sequential input and sequential output recurrent FGREP networks have essentially the same structure, and are similar to the one described in [Elman, 1988], and also used by St. John and McClelland [St. John and McClelland, 1988], and by Servan-Schreiber, Cleeremans and McClelland [Servan-Schreiber *et al.*, 1988]. 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. The previous hidden layer places the current input in context, essentially serving as sequence memory. During learning, the weights from the previous hidden layer to the hidden layer proper are modified as usual according to the backpropagation mechanism.

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. In sequential recurrent FGREP both networks are also developing representations in their input layers.

2.2 Connecting the building blocks in DISPAR

The DISPAR system (DIStributed PARaphraser) consists of two parts (figure 2). The first part reads in the story, word by word, into the internal representation, and the second part produces a word-by-word fully expanded paraphrase of the story from the internal representation. Each part consists of two sequential recurrent FGREP-modules, one for reading/producing the sentences and the other for reading/producing the words of each sentence. We

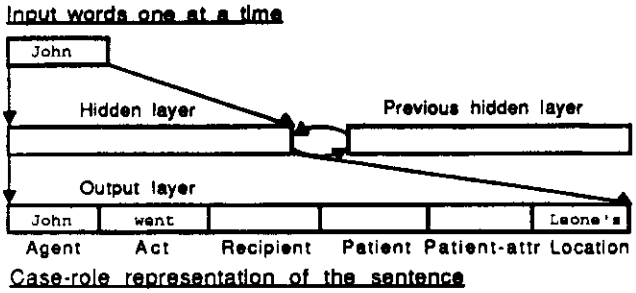

Figure 3: Word parser network.

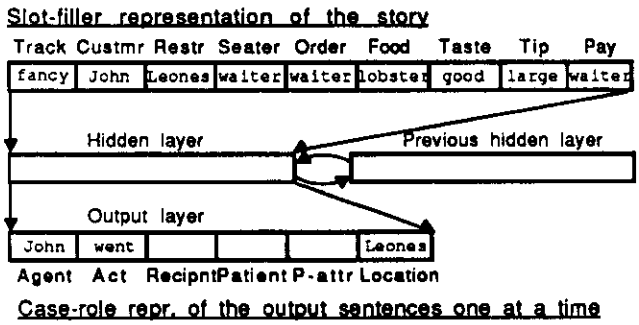

Figure 4: Event generator network.

call these modules the word parser/word generator and the event parser/event generator networks (presented in greater detail in figures 3-6). 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 consist of distributed representations of words, which are stored in a global lexicon.

2.3 Performance phase

Let us present the system with the first sentence of the story, **John went to Leone's.** The task of the word parser network is to form a stationary case-role representation of this sentence (figure 3). There is an assembly of units at the output layer for each case role, and the filler of each role is represented by the activity pattern at that assembly. The correct representation for the first sentence is agent=**john**, **act=went**, location=**Leone's**. Essentially, the word parser network is performing the *same task* as that of [McClelland and Kawamoto, 1986] and the FGREP-system [Miikkulainen and Dyer, 1988b], but *using sequential input.*

Word by word, the representations are fetched from the dictionary and loaded into the input assembly of the word parser network. 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. After a period is input, ending the first sentence, the final activity pattern is copied and used as the first input to the event

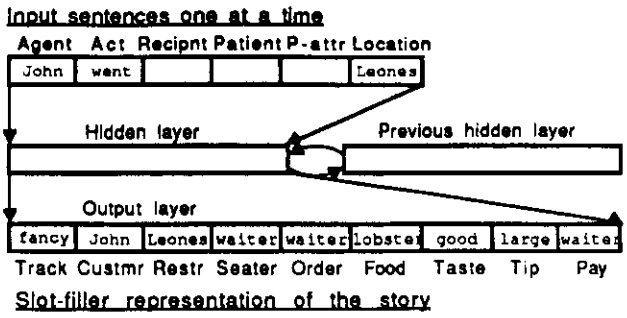

Figure 5: Event parser network.

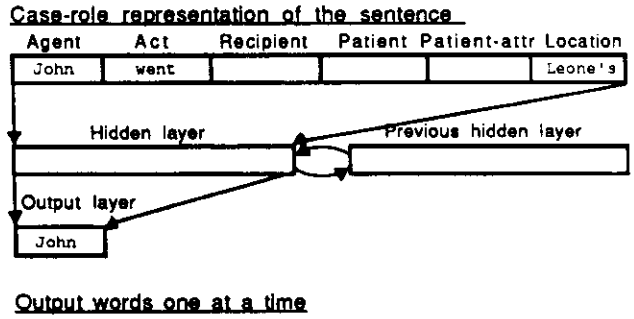

Figure 6: Word generator network.

parser network (figure 5). This network receives the case-role representations one at a time, and forms the slot-filler representation of the whole story (i.e. the script) at its output layer, in the same manner.

The final result of reading in the story is the slot-filler assignment at the output of the event parser network. In the case of our example story, track=**fancy**, customer=John, restaurant=**Leone's**, seater=**waiter**, etc. This completely specifies the events of the script, and we can train the second part of the system to paraphrase the story from this slot-filler information. The idea is simply to reverse the process of reading in.

The event generator network (figure 4) receives the complete slot-filler representation of the story as its input, and produces the case-role assignment of the first sentence of the story as its output. This output is fed into the word generator network (figure 6), which produces the distributed representation of the first word of the first sentence as its output. Again, the hidden layer of the word 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 event generator network is copied into its previous-hidden-layer assembly, and the event generator network produces the case-role representation of the second sentence. The process is repeated until the whole story has been output.

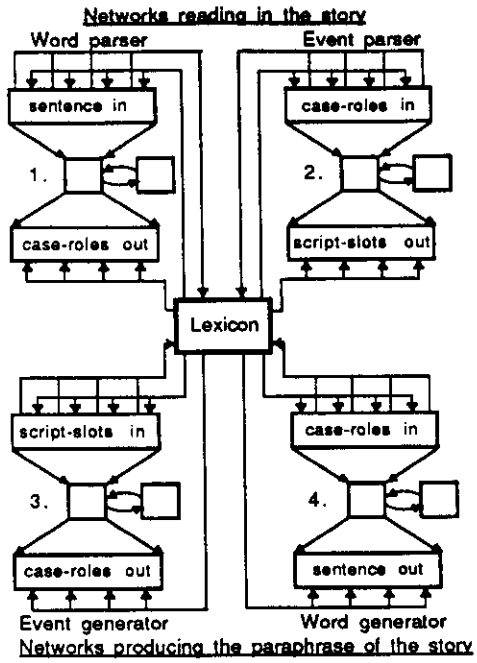

Figure 7: Training configuration.

2.4 Training phase

The four networks are trained in their tasks separately but simultaneously (figure 7). The input and the teaching patterns for each network are formed by fetching the current representations of each input and output word, case-role filler and script-slot filler from the lexicon. The activity is propagated to the output layer, error signals are formed and propagated back, changing the weights according to the backpropagation rules. At the input layer, the current input representations are changed according to the FGREP-method. The new representations of the input words are loaded back into the lexicon.

The word representations are initially all random. In the course of training, they are developed simultaneously by all four networks. Each modifies the representations to improve its performance in its task, and the representations evolve to reflect the underlying regularities of the task. When several networks are developing the same representations, the requirements of the different tasks are combined. *The resulting representations reflect the total use of the words.*

The networks are trained with compatible data. The set of teaching patterns of one network is exactly the same as the set of input patterns of another network. Even if the learning is less than complete, the networks perform well together when connected. Erroneous output patterns are simply noisy input to the next network, and neural networks in general tolerate, even filter out noise very efficiently.

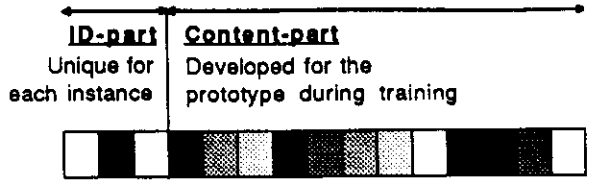

Figure 8: Representation of a word instance

2.5 Extending the vocabulary

The representations develop to reflect the underlying regularities of the task, as has been noted before and discussed at length in [Miikkulainen and Dyer, 1988a]. If two words are used in similar ways in the training data, their representations become similar. It can be argued, that the meaning of a word is manifested in how the word is used. If two words are used exactly the same, there cannot be any difference in their meaning. The representation reflects the use of the word, and therefore, can be said to encode the meaning of the word as well.

It is reasonable to assume that no two words are used exactly in similar ways in real world situations, and in principle, two words should always be distinguishable. If there is any difference in the usage, the FGREP process will develop different representations for the words. Keeping the representations separate when training an AI system with artificially generated data is problematic, though. The system should be able to handle a large and preferably open-ended set of customers, foods, restaurants, etc. without confusing them, but generating a training set which would allow enough differences to develop between their representations is unwieldy. If the paraphrasing system is allowed to develop similar representations for different customers, for example, it cannot keep role bindings straight, but alternates between different customer names when producing the story.

One way to avoid this problem is to designate a subset of the representation components for the surface level identification of the word. The representation now consists of two parts: the content part, which is developed in the FGREP process and which codes the pragmatics of the word, and the ID part, which is unique for each instance of the same word type (figure 8).

In the training phase, only a single *prototype* of role words like **customer**, **food**, and **restaurant** is developed. The network is trained to accept any ID-part. The first twenty percent of the representation units of the prototype words are set up randomly for each story, and the network is required to produce the same pattern in its output role assemblies.

In the performance phase, *it is now possible to create new instances of customers, food, and restaurants* simply by concatenating a unique ID with the content part of the developed prototype. The number of, for instance, customers the system can handle is unlimited in principle, but the role binding becomes increasingly weaker when the IDs become more similar.

The same technique can be extended to all words in the training data. In this case the network is trained only with word types (e.g. **customer**, versus, say, John). In the performance phase, several pragmatically equivalent words can be used to instantiate the word type. This technique allows us to tremendously increase the size of the vocabulary while having only a small number of pragmatically different words at our disposal. Even though in principle each word has a unique meaning (i.e. each word is its own type), this allows us to *approximate* the meaning of a large number of words by dividing them into equivalence classes.

3 Experiments

3.1 Training data

The stories were generated from the following three skeletons. These stories represent three tracks of the restaurant script: fancy-restaurant, coffee-shop, and fast-food.

Fancy-restaurant track :

```
Customer went to fancy-restaurant.
Waiter seated customer.
Customer asked the waiter for fancy-food.
Waiter brought customer the fancy-food.
Customer ate the fancy-food.
The fancy-food tasted good.
Customer paid the waiter.
Customer gave a large tip.
Customer left fancy-restaurant.
```

Coffee-shop track :

```
Customer went to coffee-shop.
Customer seated customer.
Customer asked the waiter for coffee-shop-food.
Waiter brought customer the coffee-shop-food.
Customer ate the coffee-shop-food.
The fancy-food tasted good/bad.
Customer paid the cashier.
Customer gave a large/small tip.
Customer left coffee-shop.
```

Fast-food track :

```
Customer went to fast-food-restaurant.
Customer asked the cashier for fast-food.
Cashier brought customer the fast-food.
Customer paid the cashier.
Customer seated customer.
Customer ate the fast-food.
The fast-food tasted bad.
Customer gave no tip.
Customer left fast-food-restaurant.
```

The event sequences are somewhat different in different tracks. Notice that there are certain regularities: the food is always good in a fancy-restaurant, and always bad in a fast-food restaurant. If the food is good, the size of the tip is large, and if it is bad, small, except in the fast-food restaurant where no tip is given. The system should be able to use these dependencies to fill in unmentioned roles.

Each sentence in the input ends with a period, and the network is trained to produce a period at the end of each sentence. The period is used to segment the input and the output.

3.2 Implementation

Each subnetwork was trained separately on a different HP 9000/350 workstation, each accessing the representation file over the network. It is essential that all the networks are trained simultaneously, so that the final representations reflect the processing requirements of all networks. On the other hand, the number of epochs trained need not be the same for all networks.

The learning is fastest if both the weights and the representations are updated on-line, i.e. after each backpropagation iteration. This becomes unwieldy when the networks are trained on separate machines, and it was necessary to use batch-mode update instead. The current representations were read at the beginning of each epoch, the gradient was accumulated throughout the epoch, and in the end the gradient was added to the current representations in the file (which might have been updated by another network in between). This technique is slower because the weights adapt to the current representations during the epoch, and the information is thrown away when new representations are read in. If a smaller learning rate is used for the representations than for the weights, the representations change more slowly and the problem is not as severe. However, the slower learning is more than offset by the speedup of using four processors in parallel. The complete process is about three times faster than on a single machine with on-line update.

The networks were trained for about two days with a 0.1 weight learning rate, and for another four days with 0.01. The learning rate for the representations was 1/10 of the weight learning rate. Word representations consisted of 15 units and each network's hidden layer of 50 units. The word parser and generator networks iterated for about 2,000 epochs while the event parser and generator networks went through 5,000 epochs at the same time. Most of the training effort was expended on preparing the system for the different IDs. If IDs had not been used and representations for different customers, foods, and restaurants had been developed separately, the same average error level would have been achieved in about two days.

3.3 Paraphrasing incomplete stories

Each output activity pattern produced by the word generator network was compared to the representations in the lexicon and the word whose representation was the nearest (in Euclidian distance of normalized vectors) was found. The sequence of these words was taken as the output story of the network.

The training corpus consisted of complete stories, and the system was trained to reproduce the 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 sentences, like the one in the Introduction. It turns out that the system performs very well in this respect. Stories of "natural" length, like our example, are paraphrased correctly to their full extent. The quality of the food (**good** or **bad**) is inferred correctly if the size of the tip is mentioned in the story, and vice versa. If there is

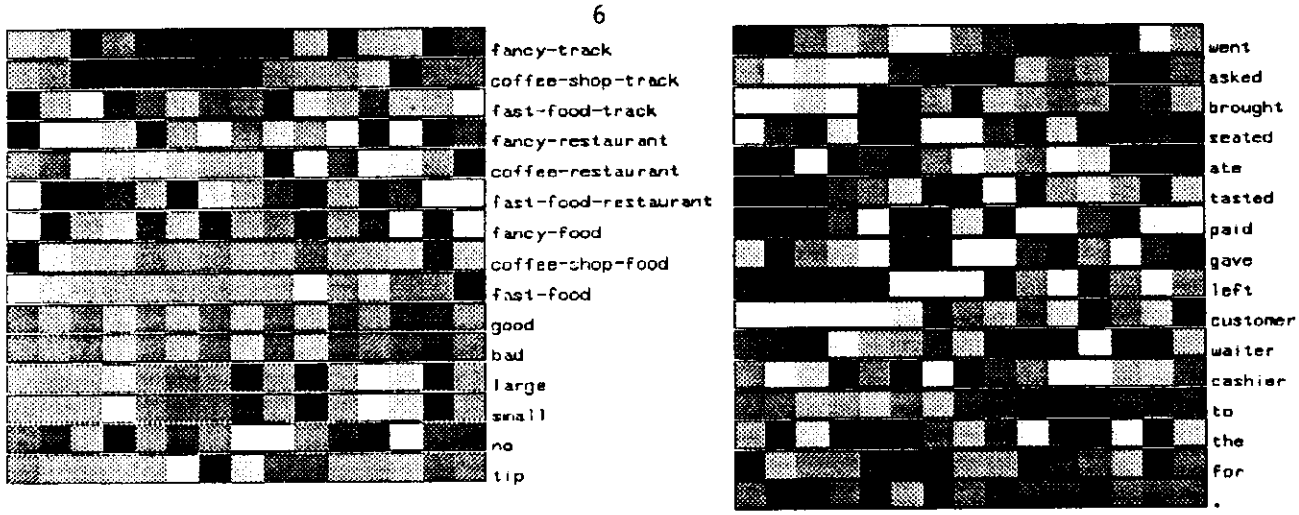

Figure 9: Final representations

not enough information to infer the filler of some role, an activity pattern is produced which is intermediate between the possible choices. For example, if the story consists of only the sentence **John went to McDonald's**, the size of the tip is inferred (no tip), and an intermediate representation develops in the food-slot. In paraphrasing the story it seems as if one of the possible fast-foods is chosen at random. But this choice is usually consistent throughout the story, because all the sentences are produced from the same pattern in the food-slot.

The strong filling-in capability of the system is partly due to the fact that there is very little variation in the training sequences. In general, a network which builds a stationary representation of a sequence may be quite sensitive to omissions and changes in the sequence. Filling in the missing items is a form of generalization. A similar generalization in a non-sequential network such as [McClelland and Kawamoto, 1986] or [Miikkulainen and Dyer, 1988b] would be required when a number of input assemblies are left blank. 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.

3.4 Representations

Some of the regularities of the stories are clearly visible in the representations that develop during training (figure 9). [**Good, bad**] and also [**large, small**] are almost identical. If these words are used symmetrically in the data (i.e. **good** always with **large**, **bad** always with **small** in all the same contexts), they actually become identical. The system can then easily produce correct output patterns for these words, but they cannot no longer be separated. Some asymmetry in their use is necessary, and we chose to use **good** always in fancy-restaurants and **bad** in fast-food-restaurants. This provides enough difference so that the words can be kept separate.

Other groups of somewhat similar words are the foods and the restaurants. There are a number of differences in the contexts where they occur, and consequently their representations become more distinct.

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 sentences are often ended prematurely with a period, and after that, only periods are output. This occurs because the representation for the period is modified at the end of every sentence, and reading it in should not change the output. Its representation evolves to the most neutral representation, i.e. it contains many components close to 0.5. This representation makes the period a default output when the network cannot handle the input.

3.5 Role binding

Prototypes were developed for the role words **customer**, **fancy-food**, **coffee-shop-food** and **fast-food**. The first three units of the representation were used as the ID. In the performance phase these were set orthogonally and the system was tested with 18 three sentence stories consisting of the entering, tipping and leaving events. Out of the 828 output words the system produced 97% correctly. The system had the most errors in the customer words, of which 91% were correct. The average error at the output layer of the each network was 0.035, 0.038, 0.030 and 0.047 (random values would give a 0.3 average error; unit activity values range between 0 and 1). Further training was still improving the performance at that point.

It is interesting to follow the operation of the system while it is reading in a story. The role bindings of a sentence (and a story as well) begin as undifferentiated pat-

terns, representing a combination of all the possible bindings at that point. When the next word is input, some of the ambiguities are removed and correct patterns are formed in the corresponding assemblies. Often the sentence representation is complete before the sentence is fully input. This occurs because the system was required to produce the complete sentence representation after each input word. The intermediate output patterns are averages of the possible complete patterns, weighted by how often they have occurred. Expectations are coded both in the weights and in the word representations.

What is especially interesting is that, once a role binding (e.g. customer=`john`) is selected in an earlier event in the script sequence, it is maintained throughout paraphrase generation. Thus, DISPAR *performs plausible role bindings* - en essential task in high-level inferencing and postulated as very difficult for PDP systems to achieve [Dyer, 1988].

4 Discussion

It would be possible to treat the story completion task as a pattern completion problem, and use a single, flat backpropagation network without teaching at the intermediate levels. The computations, however, would be very expensive and the system would not scale up. By using separate modules, we are effectively dividing a complex task into managable subgoals, which is generally a very efficient way to reduce complexity. We can read and produce a story of any length sequentially and cause the system to develop a meaningful internal representation, which can then be used by other systems, e.g. a question answering network.

The event networks contain the general semantic and scriptal knowledge which is needed for inferencing. The word networks form the specific language interface. This makes it *possible to change the function of the system in a modular fashion.* For instance, note that when a story is read into the internal representation, the only information that gets actually stored are the role bindings. The knowledge of the events in the story is in the weights of the network which paraphrases it. It is possible to train different networks to paraphrase the story from the same role bindings in a different style or detail, *even in a different language.*

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 some 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 the events which were not mentioned are often answered as if they were part of the story.

As discussed before, it seems that no two words are used exactly the same way, and consequently, no two words should have exactly the same representation. On the other hand, it seems that people also have an additional mechanism that separates the words at the sensory level if necessary. The two words sound different, and look different when printed, and people are able to use this information to keep them apart. This is what fixing a subset of the representation components achieves. We have essentially added a sensory tag for each word, which has no intrinsic meaning but it distinguishes the word from all other similar words. This tag is enough for our system to keep the words separate for role binding.

5 Future work

Pronoun reference is not a particularly hard problem in understanding script-based stories. We do not get confused when reading e.g.: **`The waiter seated him. He gave him a menu. He ordered 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 tolerate 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 phone script or robbery script occurring within a restaurant script). The simplest approach is to represent the different scripts on the same unit assemblies, very much like the different tracks are represented in the current system. An extra assembly might indicate the general class of the script (e.g. food service, travel, etc.), and the assemblies represent different roles depending on the class. We could also use distributed representations for the roles and scripts. Instantiation of a script would now have the form of a tensor product "cube" [Dolan and Smolensky, 1988]. One face of the cube stands for the script class, one for the role, and one for the filler. It is possible to represent multiple simultaneously active scripts in the same cube. Scripts can be partially activated and their boundaries become less rigid.

In a more advanced script applier we should also have mechanisms for handling script transitions, interference and distractions [Schank and Abelson, 1977]. We should be able to create new scripts and tracks in a self-organizing fashion, where often encountered sequences of events gradually become rigid stereotypical memories. The current architecture does not lend itself very easily to these extensions.

In addition to paraphrasing the story, it should be possible to develop other output functions with little modification. Answering questions about the story would be a most interesting extension. This could be inplemented as a separate network which receives as its input the slot-filler representation of the story plus the representation of the question which has been read in sequentially by the first network. This network outputs a representation of the answer sentence, which can then be output word by word using the output network. Using the previous hidden layer of the answer-producing network as the context

to the question, we could model context effects on question answering. [Lehnert, 1978]. Propagating the question to the slot-filler representation could have an effect on the ambiguous slots, i.e. the questions could modify the memory [Dyer, 1983].

6 Conclusion

We have applied sequential recurrent neural networks to a fairly high-level cognitive task, i.e. paraphrasing script-based stories. Using four hierarchically organized modular subnetworks, which are trained separately and in parallel, the complexity of the task is reduced by effectively dividing it into subgoals. The system uses sequential natural language input and output, and develops its own I/O representations for the words. The representations are stored in an external global lexicon, and they are adjusted in the course of training by all four subnetworks simultaneously, using the FGREP-method. The representations develop to improve the system's performance in the task and therefore reflect the underlying regularities of the task.

The role bindings are represented as activity patterns, while the information about the sequence of events is stored in the weights. By concatenating a unique identification with the representation developed during training, an arbitrary number of instances of the same word type can be created and used in the stories. *The system is able to infer events which are left unmentioned in the story, and make simple plausible inferences of the variable content of the story.*

References

[Bower *et al.*, 1979] G. H. Bower, J. B. Black, and T. J. Turner. Scripts in text comprehension and memory. *Cognitive Psychology*, 11:177–220, 1979.

[Cullingford, 1978] R. E. Cullingford. *Script Application: Computer Understanding of Newspaper Stories.* Technical Report 116, Yale University, Department of Computer Science, 1978. Ph.D. dissertation.

[Dolan and Smolensky, 1988] 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 Second Connectionist Models Summer School*, Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1988.

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

[Dyer, 1988] Michael G. Dyer. *Symbolic NeuroEngineering for natural language processing: a multilevel research approach.* Technical Report UCLA-AI-88-14, Computer Science Department, University of California, Los Angeles, 1988. To appear in J. Barnden and J. Pollack (Eds.) *Advances in Connectionist and Neural Computation Theory*, Ablex Publ. (in press).

[Elman, 1988] Jeffrey L. Elman. *Finding Structure in Time.* Technical Report 8801, Center for Research in Language, University of California, San Diego, 1988.

[Lehnert, 1978] Wendy G. Lehnert. *The Process of Question Answering.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1978.

[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 and Dyer, 1988a] 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 Second Connectionist Models Summer School*, Morgan Kaufmann Publishers, Inc., kaus-addr, 1988.

[Miikkulainen and Dyer, 1988b] 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.

[Rumelhart *et al.*, 1986] 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.

[Servan-Schreiber *et al.*, 1988] David Servan-Schreiber, Axel Cleeremans, and James L. McClelland. *Learning Sequential Structure in Simple Recurrent Networks.* Technical Report, Computer Science Department, Carnegie-Mellon University, 1988.

[St. John and McClelland, 1988] 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 Second Connectionist Models Summer School*, Morgan Kaufmann Publishers, Inc., Los Altos, CA, 1988.