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

A NEURAL NETWORK MODEL OF SCRIPT PROCESSING AND MEMORY

Risto Miikkulainen

April 1990
CSD-900011

A Neural Network Model of Script Processing and Memory

Risto Miikkulainen

March 1990

Technical Report UCLA-AI-90-03

A NEURAL NETWORK MODEL OF SCRIPT PROCESSING AND MEMORY [*]

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

Abstract

DISCERN is a large-scale AI system built from distributed neural networks. It reads short narratives about stereotypical event sequences, stores them in episodic memory, generates fully expanded paraphrases of the narratives, and answers questions about them. Processing is based on hierarchically organized backpropagation modules, communicating through a central lexicon of word representations. The lexicon is a double feature map, which transforms the physical word symbol into its semantic representation and vice versa. The episodic memory is a hierarchy of feature maps, where memories are stored "one-shot" at different locations. Several high-level phenomena emerge automatically from the special properties of distributed neural networks. DISCERN plausibly infers unmentioned events and unspecified role fillers, and exhibits plausible lexical access errors and memory interference behavior. Word semantics, memory organization and the appropriate script inferences are extracted from examples.

1 Introduction

Scripts [Schank and Abelson, 1977] are schemas of often encountered, stereotypical event sequences, such as visiting a restaurant, traveling, shopping etc. Each script divides further into tracks, or established minor variations. A script can be represented as a causal chain of events with a number of open roles. Script-based understanding means reading a script-based story, identifying the proper script and track, and filling its roles with the constituents of the story. Events and role fillers which were not mentioned in the story but are part of the script can now be inferred. Understanding is demonstrated by generating an expanded paraphrase of the original story, and by answering questions about the story.

DISCERN (DIstributed SCript processing and Episodic memoRy Network), is a neural network model of script-based story understanding. The primary motivation for this work is (1) to demonstrate that neural networks can be used to build a large-scale AI system, which performs at the level of symbolic AI models, and (2) to show that the special properties of distributed neural networks, such as learning from examples, automatic generalization and graceful degradation, can be put to use in a high-level cognitive task. More specific goals are (3) to account for learning the appropriate inferences for script processing from experience, (4) to show how the hierarchical structure of scripts, tracks and role bindings can be extracted from examples and used to organize the story memory, and (5) to show how word semantics can be extracted from examples of word use, and coded automatically into distributed representations.

The following is an I/O example of DISCERN. The input stories are based on the fancy-restaurant, plane-travel and electronics-shopping tracks:

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

John went to LAX. John checked in for a flight
to JFK. The plane landed at JFK.

John went to Radio-Shack.  John asked the
staff questions about CD-players.  John chose
the best CD-player.
```

The system reads the stories one at a time, word by word. An internal representation of each story is formed, where all the inferences are made explicit. The representations are stored in an episodic memory. The system can now answer questions about the stories:

```
What did John buy at Radio-Shack?
John bought a CD-player at Radio-Shack.
Where did John fly to?
John flew to JFK.
How did John like the food at MaMaison?
John thought the food was good at MaMaison.
```

After reading each question, the appropriate story representation is retrieved from the episodic memory, and the answer is generated word by word. DISCERN also generates full paraphrases of the stories. For example, it generates the expanded version of the restaurant story:

```
John went to MaMaison.  The waiter seated John.
The waiter gave John the menu.  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 left a
big tip.  John left MaMaison.
```

[*]This research was supported in part by an ITA Foundation grant and by fellowships from the Academy of Finland, the Emil Aaltonen Foundation, the Foundation for the Advancement of Technology and the Alfred Kordelin Foundation (Finland). The simulations were carried out in part on the Cray Y-MP8/864 at the San Diego Supercomputer Center.

The answers and the paraphrase show that DISCERN has made a number of inferences beyond the original story. For example, it inferred that John ate the lobster, the lobster tasted good, etc. The inferences are not based on specific rules but are statistical and learned from experience. The system has seen a number of similar stories in the past and the unmentioned events and role bindings have occurred in most cases. They are assumed immediately and automatically, and have become part of the memory about the story. In a similar fashion, human readers often confuse what was mentioned in the story with what was only inferred [Bower *et al.*, 1979].

The system extracts the appropriate inferences automatically, based on statistical correlations in the input examples. This differs from the symbolic models of script processing, where the inferences are based on handcrafted rules and representations of the script [Schank and Abelson, 1977; Cullingford, 1978].

Story processing is a higher level task than what neural networks have been applied to before, and in this work, a number of new network mechanisms for dealing with the complexity and structure of higher-level cognitive modeling are proposed. The general approach is to build from hierarchically organized modules which communicate through a central lexicon.

2 Overview of DISCERN

DISCERN can be divided into parsing, generating, question answering and memory subsystems, two modules each (figure 1). Each module is trained in its task separately and in parallel. During performance, the modules form a network of networks, each feeding its output to the input of another module.

The sentence parser reads the input words one at a time, and forms a representation of each sentence. The story parser combines the sequence of sentences into an internal representation of the story, which is then stored in the episodic memory. The story generator receives the internal representation and generates the sentences of the paraphrase one at a time. The sentence generator outputs the word sequence for each sentence.

The cue former receives a question representation, built by the sentence parser, and forms a cue pattern for the memory, which retrieves the appropriate story representation. The answer producer receives the question and the story and generates an answer representation, which is output word by word by the sentence generator.

3 Lexicon

The input/output of DISCERN consists of physical representations of words. These stand for the phonological or orthographic symbols, i.e. the phoneme strings or visual patterns of written characters. The physical representations remain fixed throughout the training and performance.

Communication within the system is based on semantic

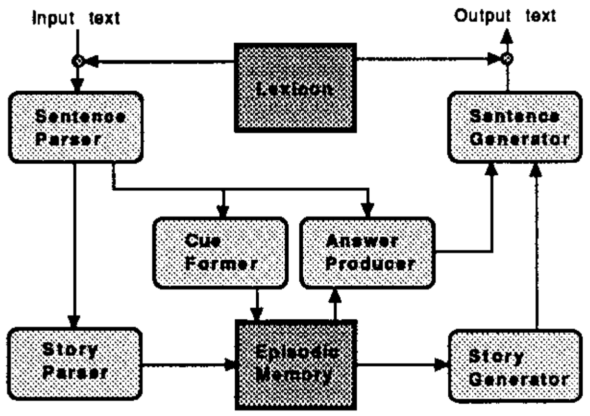


Figure 1: **Block diagram of DISCERN (performance configuration).** A dark square indicates a memory module, a light square indicates a processing module.

representations, which stand for the meanings of words. They encode how the words are used in different contexts. The semantic representations are developed by the system while it is learning the processing task.

The lexicon stores the physical and semantic representations and translates between them [Miikkulainen, 1990a] (figure 2). Both the physical and semantic words are represented distributively, i.e. as vectors of gray-scale values between 0 and 1 . In the lexicon, each high-dimensional representation space is laid out on a 2-D topological feature map, and the two representations of each word are connected.

A 2-D topological feature map [Kohonen, 1984] consists of an array of processing units, each with N weight parameters. Each unit produces one output value, proportional to the similarity of the map's current input vector and the unit's weight vector. The total response of the map is a localized pattern of activity (figure 2). In other words, an N-D input vector is mapped onto a location in the 2-D map. The weight vectors are tuned to specific items of the input space so that topological relations are retained: nearby vectors in the input space are mapped onto nearby units in the map. The organization of the map, i.e. the assignment of weight vectors, is formed in a self-organizing process, by presenting the map with randomly drawn examples of input vectors and gradually changing the weight vector values.

The lexicon is implemented as two feature maps, physical and semantic. Words whose physical forms are similar, e.g. `house`, `mouse` are represented by nearby units in the physical map. In the semantic map, words with similar semantic content, e.g. `girl`, `boy` are mapped near each other.

The two maps are densely interconnected with winner-take-all -type associative connections. A localized activity pattern representing a word in one map will cause a localized activity pattern to form in the other map, representing the same word (figure 2). The lexicon thus transforms a

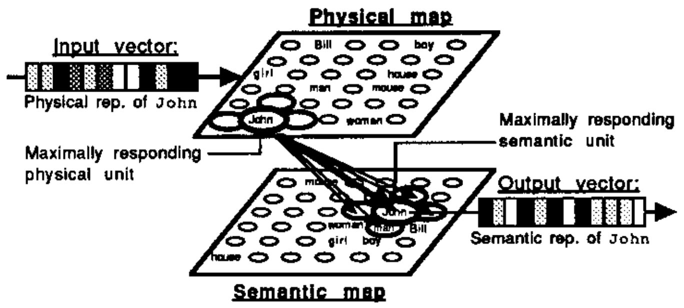
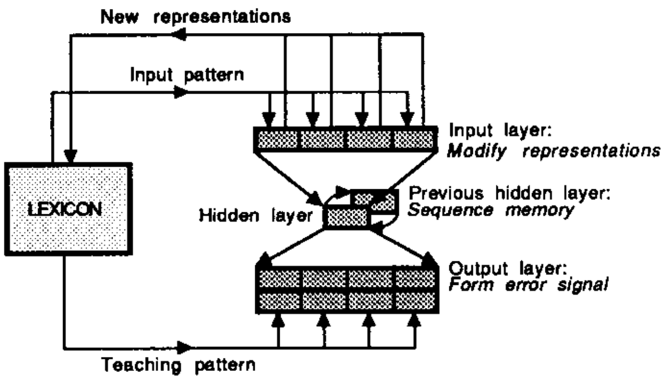


Figure 2: **The lexicon.** The physical input word "John" is transformed into the semantic representation of **John**. The size of the unit indicates the strength of its response. Only the excitatory connections from physical to semantic **John** are shown.



New representations
Input pattern
LEXICON
Input layer: Modify representations
Hidden layer
Previous hidden layer: Sequence memory
Output layer: Form error signal
Teaching pattern

Figure 3: **The FGREP module.** With sequential input or output, 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.

physical input vector into a semantic output vector, and vice versa. Both maps are organized and the associative connections between them are formed simultaneously, at the same time as the whole script paraphrasing system is being trained.

The lexicon architecture facilitates interesting behavior. Localized damage to the semantic map results in category-specific lexical deficits similar to human aphasia [Miikkulainen, 1990a]. Dyslexic performance errors can also be modeled. If the performance is degraded e.g. by adding noise to the connections, two types of parsing errors and two types of generation errors occur. In parsing, (1) a physical input pattern may be mapped incorrectly to a nearby unit in the physical map. This corresponds to reading (or hearing) the word incorrectly. Or, (2) activity in the physical map may be propagated incorrectly to a nearby unit in the semantic map. E.g. `lion` is understood semantically as `tiger`. Analogously in generation, (1) a semantic input pattern can be recognized incorrectly, and a word with a similar but incorrect meaning is produced. Or, (2) activity may be propagated to an incorrect unit in the physical map, and a word with a similar surface form but different meaning is output.

Using associative connections it is also possible to implement multiple meanings for the same physical word (homonyms), and have several different physical words correspond to one semantic word (synonyms). Priming information from the context is necessary to make a choice between alternatives. This has not been implemented in the model, but it is an interesting direction for future development [Miikkulainen, 1990a].

4 FGREP -processing modules

The processing in DISCERN is carried out by hierarchically organized FGREP modules. Each module performs a specific subtask, such as parsing a sentence or generating an answer to a question. All these modules have the same basic architecture.

The FGREP mechanism (Forming Global Representations with Extended backPropagation) [Miikkulainen and Dyer, 1989a] is based on a basic three-layer backward error propagation network, with the I/O representation patterns stored in an external lexicon (figure 3). The input and output layers of the network are divided into assemblies. A routing network forms each input pattern and the corresponding teaching pattern by concatenating the semantic lexicon entries of the input and teaching items.

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 end of each cycle, the current input representations are modified at the input layer based on the error signal. The modified representations are put back to the lexicon, replacing the old ones and thereby changing the next teaching pattern for the same input. In other words, backpropagation is shooting at a moving target in a reactive training environment.

The representations that result from this process have a number of interesting properties [Miikkulainen and Dyer, 1989a; Miikkulainen and Dyer, 1989b]. Since they adapt to the error signal, the representations end up coding properties most crucial to the task. Representations for words which are used in similar ways in the examples become similar. Thus these profiles of continuous activity values can be claimed to code the meanings of the words as well.

Single representation components do not usually stand for identifiable microfeatures. Instead, the representation is extremely holographic. Word categories can often be recovered from the values of single components, making the system very robust against damage. Performance degrades approximately linearly as components become defective.

The representation for a word is determined by all contexts where that word has been encountered, and consequently, it is also a representation of all these contexts. Expectations emerge automatically and cumulatively from the input word representations. Also, the system never has to process very novel input patterns, because generalization has already been done in the representations.

Three types of FGREP modules are used in the system: non-recurrent (the cue former and the answer producer),

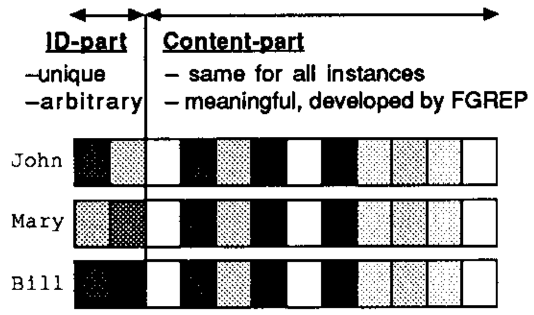


Figure 4: **Cloning word instances.** Instances **John**, **Mary** and **Bill** are created from the prototype word **human**.

sequential input (the parsers), and sequential output modules (the generators). In recurrent modules the previous hidden layer serves as sequence memory, remembering where in the sequence the system currently is and what has occurred before [Elman, 1988] (figure 3). In a sequential input network, the 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 input is stationary, but the teaching pattern changes at each step. The network is producing a sequential interpretation of its input.

It is possible to extend the FGREP vocabulary by creating a number of distinct word instances from the same semantic word, e.g. tokens **John**, **Mary**, **Bill** from the type **human** (figure 4). The representation now consists of two parts: the content part, which was developed by 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. The ID part has no intrinsic meaning in the system, but it distinguishes the word from all other instances of the same word. The technique can be thought of as an approximation of sensory grounding, where the ID part stands for the sensory referent of the word.

The ID technique can be applied to any word in the training data, and in principle, the number of instances per word is unlimited. This allows us to approximate a large vocabulary with only a small number of semantically different representations at our disposal. Word discrimination degrades approximately linearly as a function of instances, which is remarkable since the number of different input/output patterns grows polynomially [Miikkulainen and Dyer, 1989b].

5 Episodic memory

The episodic memory is a hierarchical feature map system [Miikkulainen, 1990b] combined with the trace feature map mechanism [Miikkulainen, 1990c]. The map hierarchy provides the organization for the memory, and the trace map technique is used to implement the memory traces.

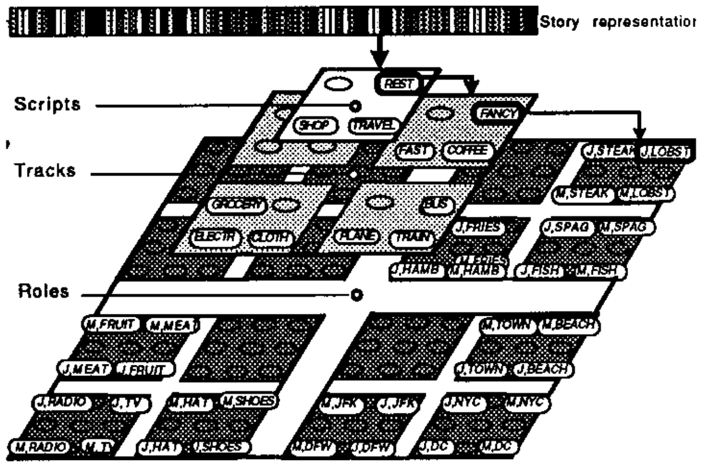


Figure 5: **The hierarchical feature map classification of script-based stories.** Labels indicate the maximally responding unit for each script, track and role binding.

5.1 The feature map hierarchy

The feature map hierarchy is a pyramid organized according to the hierarchical taxonomy of script-based stories (figure 5). The highest level of the hierarchy is a single feature map, which lays out the different script classes. Beneath each unit of this map there is another feature map, which lays out the tracks within the script. At the bottom level, the different role bindings within each track are separated. The map hierarchy receives a story representation as its input and classifies it as an instance of a particular script, track and role binding. In other words, the map hierarchy provides a unique memory representation for each script-based story.

Let us follow the classification of John's visit to MaMaison. The top-level map receives the complete representation vector and maps it onto the unit labeled **REST** (figure 5). This unit compresses the vector by removing the components whose values are the same in all restaurant stories. The representation now consists of information which best distinguishes between the different restaurant stories. The **REST**-unit passes the compressed representation down to its submap, which classifies it as an instance of the fancy-restaurant track. Again, the **FANCY**-unit removes the components common to all fancy-restaurant stories, and passes the highly compressed vector to its submap. The representation is now limited to information about the role bindings, and it is mapped onto the unit representing customer=**John**, food=**lobster**.

A higher-level map in the hierarchy acts as a filter, (1) choosing the relevant input items for each lower-level map and (2) compressing the representation of these items to the most relevant components. The maps lower in the hierarchy form finer and finer distinctions between the stories.

The hierarchical script taxonomy is extracted from examples of story representations. The pyramid structure is predetermined and fixed, and the maps are self-organized one level at a time from top to bottom. Each unit inde-

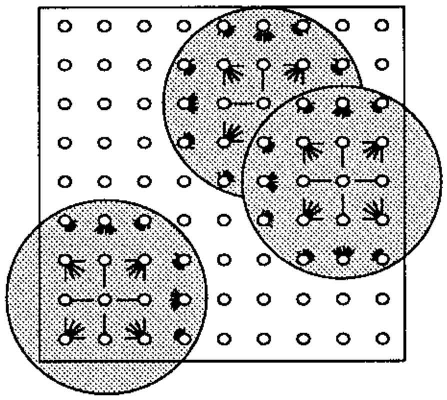


Figure 6: **A trace feature map.** Line segments indicate positive lateral weights originating from each unit. The trace on the right has partially obscured an earlier trace.

pendently determines how to compress its input vectors, by finding the components with the least variance. The organizing process is very efficient, because it employs hierarchical subgoals. The maps are small at all levels, and they receive only selected, condensed input vectors.

Hierarchical feature maps have a number of properties which make them useful for memory organization: (1) the maps visualize the data very nicely, with the hierarchy displaying the script taxonomy and the maps laying out the topology at each level, (2) the most salient aspects of the input data are separated and most resources are concentrated on them, (3) the organization is formed in an unsupervised manner, extracting it from input examples, (4) the classification is very robust, and usually correct even if the input vector is noisy or incomplete.

5.2 Trace feature maps

An ordinary feature map is a classifier, mapping an input vector onto a location on the map. A trace feature map, in addition, creates a memory trace on that location. The map remembers that at some point it received an input item that was classified there. The traces can be stored one at a time, as stories are read in, and retrieved with a partial cue.

A trace feature map is a single ordered feature map, with modifiable lateral connections between units (figure 6). Initially the lateral connections are all inhibitory. When an input vector is presented to this map, a localized activity pattern forms as a response. A trace is created by modifying the lateral connections of the units within this response. A connection to a unit with a higher activity is made excitatory, while a connection to a unit with a lower activity is made inhibitory, both proportional to the activity level of the source unit. The units within the response are now 'pointing' towards the unit with the highest activity (figure 6).

A stored vector is retrieved by presenting the map with an approximation of the vector. The initial response is again a localized activity pattern. If the response is close enough to a stored trace, the lateral connections pull the activity to the center of the trace, and the weights of this unit give to the stored vector. If the cue is too far, the response does not reach the 'basin' of the trace, and the activity oscillates between nonactivity (caused by the inhibitory lateral connections) and the initial response. In other words, the trace map can complete a partial cue, and indicate when there is no appropriate trace in the memory.

The trace map exhibits interesting memory effects which result from interactions between traces. Later traces steal units from earlier ones, making later traces more likely to be retrieved (figure 6). The extent of the basins determines the memory capacity. The smaller the basins, the more traces will fit in the map, but more accurate cues are required to retrieve them. If the memory capacity is exceeded, older traces will be selectively replaced by newer ones. Traces which are unique, i.e. in a sparse area of the map, are not affected, no matter how old they are.

5.3 Storage and retrieval

The episodic memory represents a story by the maximally responding units at each level. However, a trace needs to be created only at the bottom level. The script and the track level are ordinary feature maps, while the role binding level consists of trace feature maps.

When a representation is stored in the episodic memory, the map hierarchy determines the appropriate role binding map and the location on that map. The trace feature map mechanism then creates a memory trace at that location.

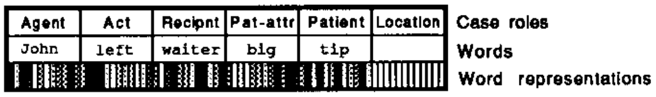
A story is retrieved from the memory by giving it a partial story representation as a cue. Unless the cue is highly deficient, the map hierarchy is able to recognize it as an instance of the correct script and track, and form a partial cue to the role binding map. The trace map mechanism then completes the role binding. The complete story representation is retrieved from the weight vectors of the maximally responding units at the script, track, and role binding levels.

6 Connecting the modules in DISCERN

6.1 Performance phase

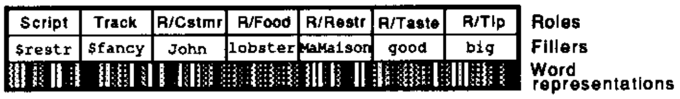
Let us follow DISCERN (figure 1) as it processes the story about John's visit to MaMaison. The physical representations for each word are presented to the physical map of the lexicon, which produces the corresponding semantic representation as its output (figure 2). These are fed one at a time to the sentence parser, which gradually forms a stationary case-role representation of each sentence (figure 7) at its output layer. After a period is input, ending the sentence, the final case-role pattern is copied to the input of the story parser.

In a similar manner, the story parser receives a sequence of sentence case-role representations as its input, and forms a stationary slot-filler representation of the whole story (figure 8) at its output layer. This is a representation of the story in terms of its role bindings, and constitutes the



Agent	Act	Recipnt	Pat-attr	Patient	Location	Case roles
John	left	waiter	big	tip		Words
						Word representations

Figure 7: **Case-role representation of the sentence John left the waiter a big tip.** The word representations equal the semantic lexicon representations.



Script	Track	R/Cstmr	R/Food	R/Restr	R/Taste	R/Tip	Roles
$restr	$fancy	John	lobster	MaMaison	good	big	Fillers
							Word representations

Figure 8: **Representation of the story by its role bindings.** The assemblies are not role-specific, but their interpretation depends on the pattern in the script and track slots. The role names R/... are specific for the restaurant script.

final result of the parse.

The story representation is fed to the episodic memory, which classifies it as an instance of a script, track, and role binding and creates a trace in the appropriate trace map (figure 5).

The generator subsystem reverses the parsing process. The story generator network receives the story representation as its input and generates a sequence of sentence case-role representations. Each of these is fed to the sentence generator, which outputs the semantic representations of the output words one at a time. Finally, the lexicon transforms these into physical words.

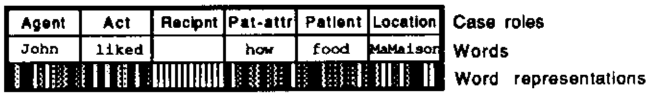
The sentence parser and the sentence generator are also trained to process question sentences and answer sentences. The cue former takes the case-role representation of the question (figure 9), produced by the sentence parser, and generates a partial story representation as its output (figure 10). This pattern is fed to the episodic memory, which classifies it as an instance of a script, track, and role binding. The trace map settles to a previously stored memory trace, and the complete story representation (figure 8) is retrieved from the weights of the maximally responding units.

The answer producer receives the complete story representation, together with the case-role representation of the question, and generates a case-role representation of the answer sentence (figure 11), which is then output word by word by the sentence generator.

6.2 Training phase

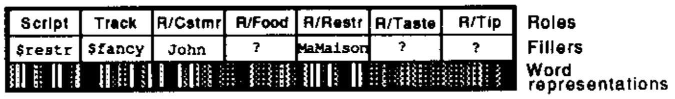
A good advantage of the modular architecture can be made in training the system (figure 12). The tasks of the six processing modules are separable, and they can be trained separately as long as compatible I/O material is used. The modules must be trained simultaneously, so that they develop and learn to use the same semantic representations. The hierarchical organization of the episodic memory can be developed at the same time.

The lexicon ties the separated tasks together. Each FGREP network modifies the representations to improve its performance in its own task. The pressure from other



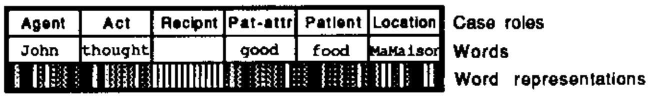
Agent	Act	Recipnt	Pat-attr	Patient	Location	Case roles
John	liked		how	food	MaMaison	Words
						Word representations

Figure 9: **Case-role representation of the question How did John like the food at MaMaison?** Questions are represented as sentences, but processed through different pathways.



Script	Track	R/Cstmr	R/Food	R/Restr	R/Taste	R/Tip	Roles
$restr	$fancy	John	?	MaMaison	?	?	Fillers
							Word representations

Figure 10: **Memory cue.** Most of the story representation is complete, but the patterns in Food, Taste and Tip slots indicate averages of all possible alternatives.



Agent	Act	Recipnt	Pat-attr	Patient	Location	Case roles
John	thought		good	food	MaMaison	Words
						Word representations

Figure 11: **Case-role representation of the answer John thought food was good at MaMaison.**

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.

After training is complete, 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 efficiently.

7 Discussion

The complete system performs very well in the task. Missing events and role fillers are plausibly inferred, and at the output, about 99% of the words are correct (tested with 72 three sentence stories, instantiated from three scripts, each with three tracks, each with three open roles, and two instances cloned for each filler word). If there is not enough information to fill some role, the most likely role binding is selected and maintained throughout the paraphrase generation. Thus, DISCERN performs plausible role bindings – an essential task in high-level inferencing and postulated as very difficult for PDP systems to achieve [Dyer, 1989].

The episodic memory for the script processing system has two requirements, which are not met by most neural network models of associative memory, e.g. [Kohonen, 1984; Hopfield, 1982]: (1) memories need to be stored one-shot, with only a single presentation, without knowledge about future memories to be stored, and (2) a few components in the representation, i.e. the IDs, need to be stored and retrieved with extreme accuracy. The first problem was solved with the trace map mechanism by using dif-

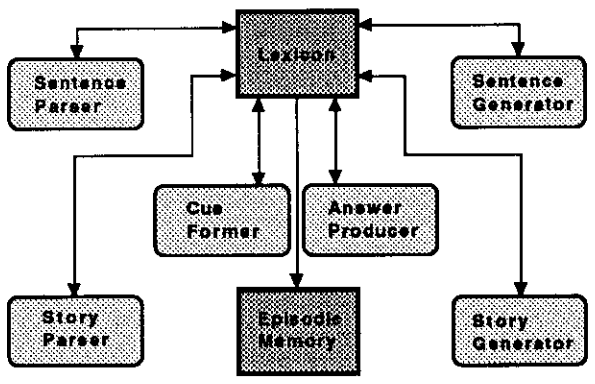


Figure 12: **Training configuration.** Each module is trained separately and simultaneously with compatible I/O data, developing the same lexicon.

ferent areas of the network to store different memories. The solution to the second problem was provided by the hierarchical feature maps, which employ abstractions to separate crucial information (IDs, mapped at the bottom level) from information which is more widely distributed and where accuracy is not as critical (components common to the script and track, at higher levels).

The connectionist models typically have had 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 modeling well-defined, isolated low-level tasks, such as learning past tense forms or the pronunciation of words. Our results suggest, in the spirit of [Minsky, 1985], that a plausible approach for higher level cognitive modeling is to compose the model from several simple submodules, which work together to produce the higher level behaviour.

8 Conclusion

The distributed neural network approach is quite effective in script processing. Scripts are regular event sequences, and their structure can easily be extracted by a neural network system. Script inferences are intuitive, immediate and occur without conscious control, a process which automatically arises from generalization and graceful degradation in distributed networks. Also significantly, DISCERN processes both semantic (general, statistical) and episodic (specific, one-shot) information and is able to produce sequential high-level behaviour. We see this as a promising beginning in tackling complex cognitive tasks with neural networks.

References

[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.

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

[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).

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

[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.

[Kohonen, 1984] Teuvo Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, Berlin; New York, 1984.

[Miikkulainen, 1990a] Risto Miikkulainen. *A Distributed Feature Map Model of the Lexicon.* Technical Report UCLA-AI-90-04, Artificial Intelligence Laboratory, Computer Science Department, University of California, Los Angeles, 1990.

[Miikkulainen, 1990b] Risto Miikkulainen. Script recognition with hierarchical feature maps. *Connection Science*, 1990. (In press).

[Miikkulainen, 1990c] Risto Miikkulainen. Trace feature map: A one-shot learning associative memory. 1990. In preparation.

[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. Natural language processing with modular neural networks and distributed lexicon. 1989. Submitted to *Cognitive Science.*

[Minsky, 1985] Marvin Minsky. *Society of Mind.* Simon and Schuster, New York, 1985.

[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.