

# Script-Based Inference and Memory Retrieval in Subsymbolic Story Processing <sup>\*†</sup>

Risto Miikkulainen  
Department of Computer Sciences  
The University of Texas at Austin, Austin, TX 78712  
risto@cs.utexas.edu

## Abstract

DISCERN is an integrated natural language processing system built entirely 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 in DISCERN is based on hierarchically-organized backpropagation modules, communicating through a central lexicon of word representations. The lexicon is a double feature map system that transforms each orthographic 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 in this model. DISCERN learns to infer unmentioned events and unspecified role fillers, generates expectations and defaults, and exhibits plausible lexical access errors and memory interference behavior. Word semantics, memory organization, and appropriate script inferences are automatically extracted from examples. DISCERN shows that high-level natural language processing is feasible through integrated subsymbolic systems. Subsymbolic control of high-level behavior and representing and learning abstractions are the two main challenges in scaling up the approach to more open-ended tasks.

## 1 Introduction

DISCERN (DIstributed SCRipt processing and Episodic memoRY Network) is a subsymbolic artificial neural network system that learns to process simple stereotypical narratives. To see what DISCERN is up against, let us consider the following input examples:

- (1) John went to MaMaison. John asked the waiter for lobster. John left a big tip.
- (2) John went to LAX. John checked in for a flight to JFK. The plane landed at JFK.
- (3) John went to Radio-Shack. John asked the staff questions about CD-players. John chose the best CD-player.

The first narrative mentions only three events about John’s visit to MaMaison. Because restaurant visits are common experiences with very regular events, a human reader immediately assumes a number of events that certainly (or most likely) must have occurred. For example, he/she assumes that the waiter seated John, that John ate the lobster, and that John paid the waiter. Based on the fact that the restaurant serves lobster, and that John left a big tip, the reader can also guess that the restaurant was probably of the fancy

---

<sup>\*</sup>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) when the author was at UCLA. The simulations were carried out in part on the Cray Y-MP8/864 at the San Diego Supercomputer Center.

<sup>†</sup>To appear in *Applied Intelligence*.

type rather than a fast-food type, and that the food was probably good. If asked to elaborate, the reader might come up with the following expanded paraphrase:

John went to MaMaison. The waiter seated John. The waiter gave John the menu. John asked the waiter for lobster. John waited for a while. 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.

Or the reader could answer questions about the stories in the following way:

Q: What did John buy at Radio-Shack?  
A: John bought a CD-player at Radio-Shack.  
Q: Where did John take a plane to?  
A: John took a plane to JFK.  
Q: How did John like the lobster at MaMaison?  
A: John thought the lobster was good at MaMaison.

The above examples illustrate a number of issues that make script-based story understanding an interesting task. The answers and the paraphrase show that the reader makes a number of inferences beyond the original story. The inferences are not based on specific rules but on statistical regularities, learned from experience. The reader has experienced a number of similar event sequences in the past and the unmentioned events have occurred in most cases. They are assumed immediately and automatically, and quickly become part of the memory about the narrative. If the reader is asked later whether John actually flew to JFK, he would quite confidently confirm this, but might not be able to tell whether this event was actually mentioned in the story or only inferred.

Another group of issues concerns episodic memory. It seems people can store narratives in memory one at a time as they are read in, without explicitly reactivating previous traces in the memory. However, the new story is recognized as an instance of a familiar sequence of events, and attention is paid only to the facts that are specific to this story. It seems that human episodic memory is structured to support classification based on similarities and storing the differences, and that this structure has been extracted from experience. The structure also supports associative retrieval. A question supplies only partial information about the story it refers to, yet the story is retrieved with only the question as a cue. The more unique the story is in the memory, the less needs to be specified in the question. If there is only one travel story, we can unambiguously ask **Where did John travel to?** Usually, if there are several alternative stories, the most recent one is recalled by default. And of course, it is possible to recognize a situation where there is nothing appropriate in the memory.

Understanding and retrieving stories about stereotypical event sequences is not a new task. Script theory [46] was developed to explain how knowledge about familiar everyday routines is used in story understanding. At the same time, symbolic computer models such as SAM [7] and FRUMP [8] were built that implemented the theory at various levels. The IPP [27] and CYRUS [26] systems showed how episodic memory can be organized based on a few fundamental schemas and a number of specific examples. These models demonstrated the power of scripts and schemas in processing texts about everyday events, and were capable of quite impressive behavior in their domains.

What makes the task worth doing with neural networks? There are several issues that the symbolic approach does not address. The processing architecture, mechanisms, and knowledge in symbolic systems are hand-coded with a particular domain and data in mind. Inferences are based on handcrafted rules and representations of the scripts. Such systems cannot utilize the statistical properties of the data to enhance processing; 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 [11; 46]. Generalization to previously unencountered inputs is possible only if there exists a rule that specifies how this is done. Representations of these rules and their application is often very complex. Such an explicit rule-based approach seems unnatural, given how immediate and low-level such operations as expectation and generalization are for people. Specifically, this contradicts the intuitive notion of scripts. They should not be preset, fixed,

all-or-none representations, but instead should emerge automatically from the statistical regularities in the experience [44; 45].

A major motivation for DISCERN is to give a better account of these high-level cognitive phenomena in terms of the special properties of subsymbolic systems, such as learning from examples, automatic generalization and graceful degradation. The approach taken in DISCERN is that of integrated subsymbolic modeling. Separate distributed neural network models of various subtasks are brought together into a single high-level system that is similar in scope to the traditional symbolic systems.

This paper concentrates on the types of reasoning that DISCERN performs in processing script-based stories. A brief introduction to the DISCERN model is first given (for a more detailed description see [34]), followed by a description of the training and testing methodology. Several specific examples of reasoning in DISCERN are then presented, including script-based inference in parsing, paraphrasing, and question answering, dealing with ambiguous, incomplete, and erroneous input, and memory confusions and forgetting. The subsymbolic approach to processing schemas is compared to the symbolic approach, and various issues in subsymbolic modeling such as the nature of subsymbolic inference, processing types and tokens, and generalization to novel and exceptional inputs are discussed. The paper concludes by proposing subsymbolic control and learning abstractions as the two key issues in building more advanced subsymbolic models of high-level reasoning.

## 2 Overview of the DISCERN architecture

DISCERN is an integrated subsymbolic system. Its components were designed as independent cognitive models that by themselves account for interesting language-processing and memory phenomena. In DISCERN, these models are further shown to be sufficient constituents for generating complex high-level behavior.

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 case-role representation of each sentence. The story parser combines the sequence of sentences into an internal slot-filler 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 sequence of words for each sentence. The cue former receives a question representation, built by the sentence parser, and forms a cue pattern for the episodic memory, which returns 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.

The sentence representation is loosely based on the theory of thematic case roles [15; 6]. The slots in the representation correspond to the shallow semantic cases such as agent, patient, recipient, and location. The story representation follows the standard representation for scripts in symbolic systems. A script is seen as a causal chain of events with a number of open roles. Each script divides further into tracks, or established minor variations. In the representation, the slots identify the script, the track, and the fillers for roles such as customer, food, restaurant, quality of food, and size of the tip.

### 2.1 Lexicon

The input and output of DISCERN consists of distributed representations for orthographic word symbols (also called lexical words below). Internally, DISCERN processes semantic concept representations (semantic words). Both the lexical and semantic words are represented distributively as vectors of gray-scale values between 0.0 and 1.0. The lexical representations are based on the visual patterns of characters that make up the written word. They remain fixed throughout the training and performance of DISCERN. The semantic representations stand for distinct meanings. They are developed automatically by the system while it is learning the processing task.

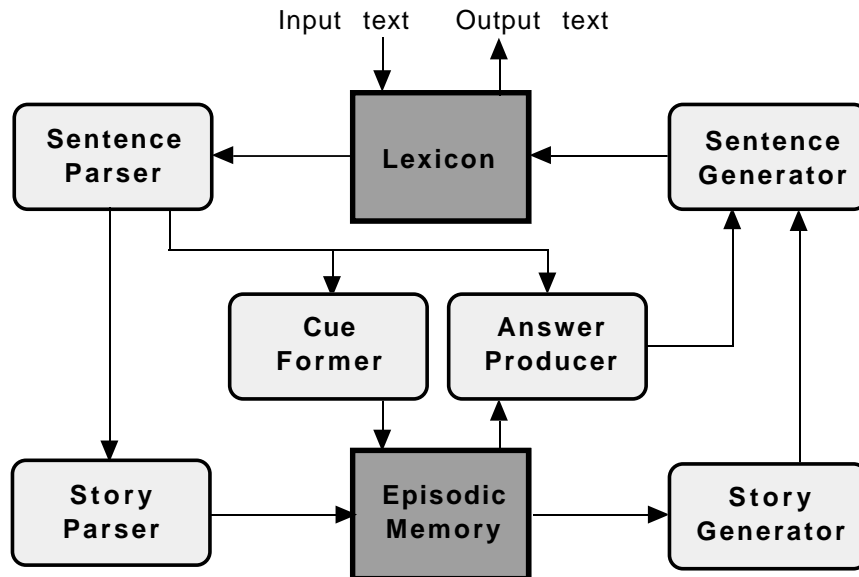


Figure 1: The DISCERN architecture (performance configuration). The model consists of parsing, generating, question answering, and memory subsystems, two modules each. A dark square indicates a memory module, a light square indicates a processing module. The lines indicate pathways carrying distributed word, sentence, and story representations during the performance phase of the system. The modules are trained separately with compatible I/O data.

The lexicon [31] stores the lexical and semantic representations and translates between them (figure 2). It is implemented as two feature maps [24; 25], lexical and semantic. Words whose lexical forms are similar, such as **LINE** and **LIKE**, are represented by nearby units in the lexical map. In the semantic map, words with similar semantic content, such as **John** and **Mary** or **Leone's** and **MaMaison** are mapped near each other.

There is a dense set of associative interconnections between the two maps. 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 output representation is then obtained from the weight vector of the most highly active unit. The lexicon thus transforms a lexical input vector into a semantic output vector, and vice versa. Both maps and the associative connections between them are organized simultaneously, based on examples of co-occurring symbols and meanings.

The lexicon architecture facilitates interesting behavior. Localized damage to the semantic map results in category-specific lexical deficits similar to human aphasia [4; 30]. For example, the system selectively loses access to restaurant names, or animate words, when that part of the map is damaged. Dyslexic performance errors can also be modeled. If the performance is degraded, for example, by adding noise to the connections, parsing and generation errors occur quite similarly to those observed in human deep dyslexia [5]. For example, the system may confuse **Leone's** with **MaMaison**, or **LINE** with **LIKE**, because they are nearby in the map and share similar associative connections.

## 2.2 FGREP processing modules

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, [36]) 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 (i.e.

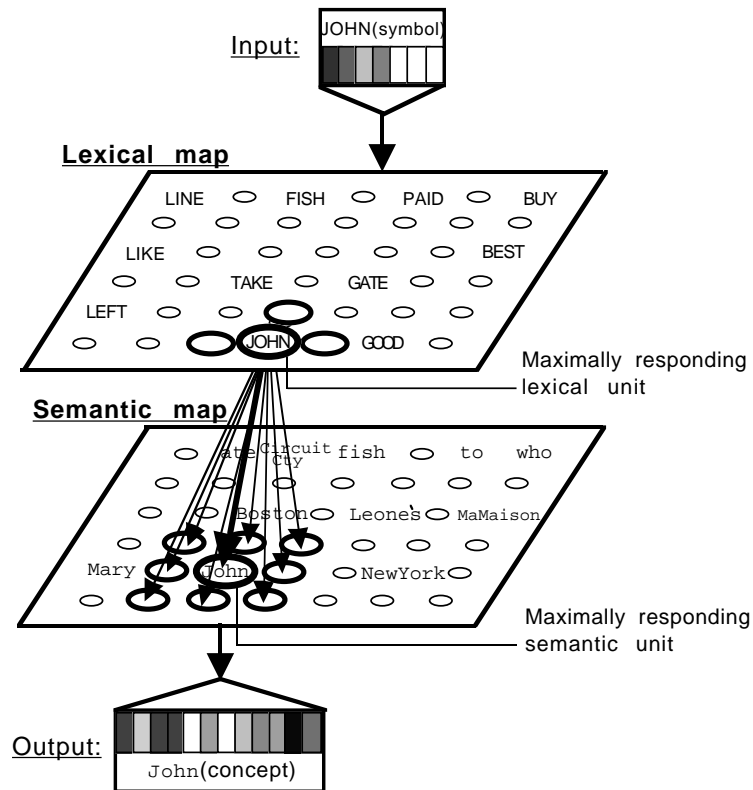


Figure 2: **The lexicon.** The lexical input symbol JOHN is translated into the semantic representation of the concept John. The representations are vectors of gray-scale values between 0.0 and 1.0, stored in the weights of the units. The size of the unit on the map indicates how strongly it responds. Only a small part of each map, and only a few strongest associative connections of the lexical unit JOHN are shown in this figure.

groups of units). The assemblies stand for slots in the I/O representation, such as the different case roles of the sentence representation, or the role bindings in the story representation. Each input pattern is formed by concatenating the current semantic lexicon entries of the input words; likewise, the corresponding target pattern is formed by concatenating the lexicon entries of the target words. For example, the target sentence **John went to Mamaison** would be represented at the output of the sentence parser network as a single vector, formed by concatenating the representations for **John**, **went**, and **MaMaison** (figure 7).

The network learns the processing task by adapting the connection weights according to the standard on-line backpropagation procedure [43, pages 327-329]. The error signal is propagated to the input layer, and the current input representations are modified as if they were an extra layer of weights (but constrained within the interval  $[0, 1]$ ):

$$r_{ci}(t+1) = \max(0, \min(1, r_{ci}(t) + \eta\delta_{1i})), \quad (1)$$

where  $r_{ci}(t)$  is the value of the representation component  $i$  of word  $c$  at time  $t$ ,  $\delta_{1i}$  is the error signal of the corresponding input layer unit, and  $\eta$  is the learning rate. The modified representation vectors are put back in the lexicon, replacing the old representations. Next time the same words occur in the input or as targets, their new representations are used to form the input/target patterns for the network. In FGREP, therefore, the required mappings change as the representations evolve, and backpropagation is shooting at a moving target.

The representations that result from this process have a number of interesting properties. Since they adapt to the error signal, the representations end up coding properties most crucial to the task. Representations for words that 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. Interestingly, single representation

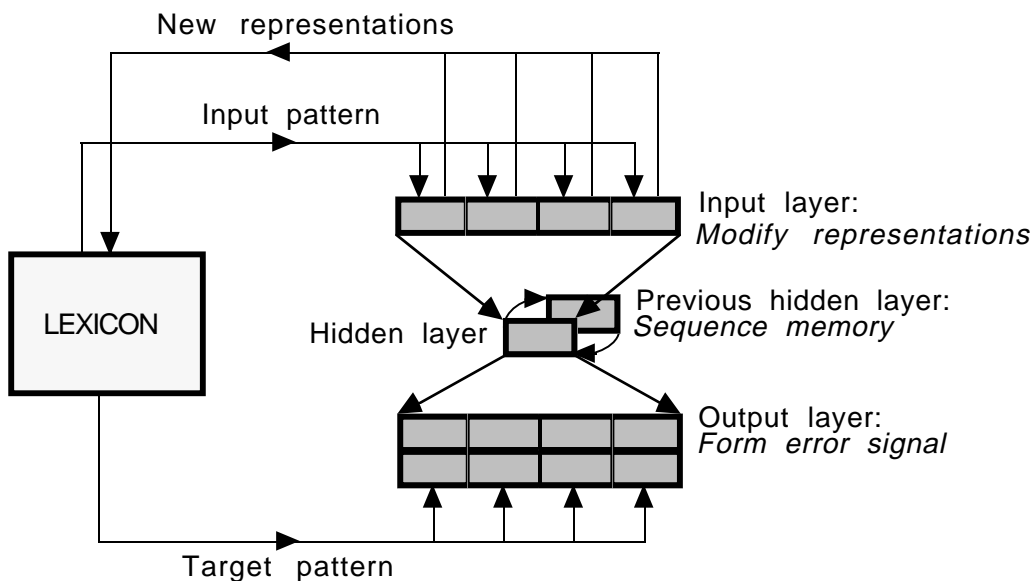


Figure 3: **The FGREP-module.** At each I/O presentation, the representations at the input layer are modified according to the backpropagation error signal, and replace the old representations in the lexicon. In the case of sequential input or output, the hidden layer pattern is saved after each step in the sequence, and used as input to the hidden layer during the next step, together with the actual input.

components do not usually stand for identifiable semantic features. Instead, the representation is 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 representation components become defective.

Three types of FGREP modules are used in the system: non-recurrent (the cue former and the answer producer), sequential input (the parsers), and sequential output modules (the generators). In the recurrent modules the previous hidden layer serves as sequence memory, remembering where in the sequence the system currently is and what has occurred before (figure 3). These networks have the same structure as Elman's Simple Recurrent Networks [13], but they are used differently. In a sequential input network, the input changes at each time step, while the teaching pattern stays the same. The network learns to form 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 learns to produce a sequential interpretation of its input.

After a core set of semantic representations have been developed in the FGREP process, it is possible to extend the vocabulary through a technique called cloning. Each FGREP representation stands for a unique meaning and constitutes a semantic prototype. In cloning, several distinct copies, or instances, are created from the same prototype. For example, instances such as **John**, **Mary**, and **Bill** can be created from the prototype **human** (figure 4). Such cloned representations consist of two parts: the content part, which was developed in the FGREP process and encodes the meaning of the word, and the ID part, which is unique for each instance of the same prototype. They all share the same meaning, and therefore the system knows how to process them, but at the same time, the ID part allows the system to keep them distinct.

The ID+content 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 (which are expensive to develop) at our disposal.

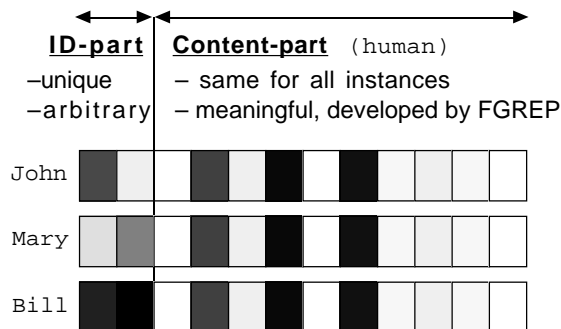


Figure 4: **Cloning word instances.** Instances *John*, *Mary*, and *Bill* are created from the prototype word *human*, which was developed in the FGREP process. The ID part has no intrinsic meaning in the system and can be assigned e.g. randomly; its task is only to distinguish the representation from all other instances of the same word prototype.

## 2.3 Episodic memory

### 2.3.1 Map hierarchy

The episodic memory in DISCERN is a hierarchical feature map system [32] combined with the trace feature map mechanism [33]. The map hierarchy provides the organization for the memory, and the trace feature map technique implements the storage and retrieval of memory traces.

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 that lays out the different script classes. Beneath each unit of this map there is another feature map that lays out the tracks within the particular 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 the story about *John's* visit to *MaMaison*. The top-level map receives the complete story representation vector and maps it onto the unit labeled *REST*, for restaurant script (figure 5). All the different restaurant stories are mapped on this same unit. The *REST*-unit “compresses” the input vector by removing all components whose values are the same in all restaurant stories. The remaining representation consists of information that best distinguishes between the different restaurant stories. The *REST*-unit passes this representation down to its submap, which has the task of classifying the restaurant stories further into tracks. In this case, the input is classified as an instance of the fancy-restaurant track. Again, the *FANCY*-unit removes the components common to all fancy-restaurant stories, and passes the remaining vector to its submap. The vector now contains information only about the role bindings, and it is mapped onto the unit representing *customer=John*, *food=lobster*, *restaurant=MaMaison*, *tip=big*.

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 itself (i.e. the number of maps at each level and the number of units in each map) is predetermined and fixed, but the system figures out how to make use of the hardware to best describe the data. The maps are self-organized one level at a time from top to bottom. Once the top-level map has learned to categorize the inputs, each of its units determines how to compress the input vectors it wins (by removing the components with the least variance), and starts passing these vectors down to its submap. The middle-level maps then form a mapping of the tracks of each script. In a similar process, each middle-level unit determines how to compress its own input vectors, and begins to pass them down to its bottom-level map. These maps develop the layout of the role bindings of each track, because that is the only information left in the input vectors at this level.

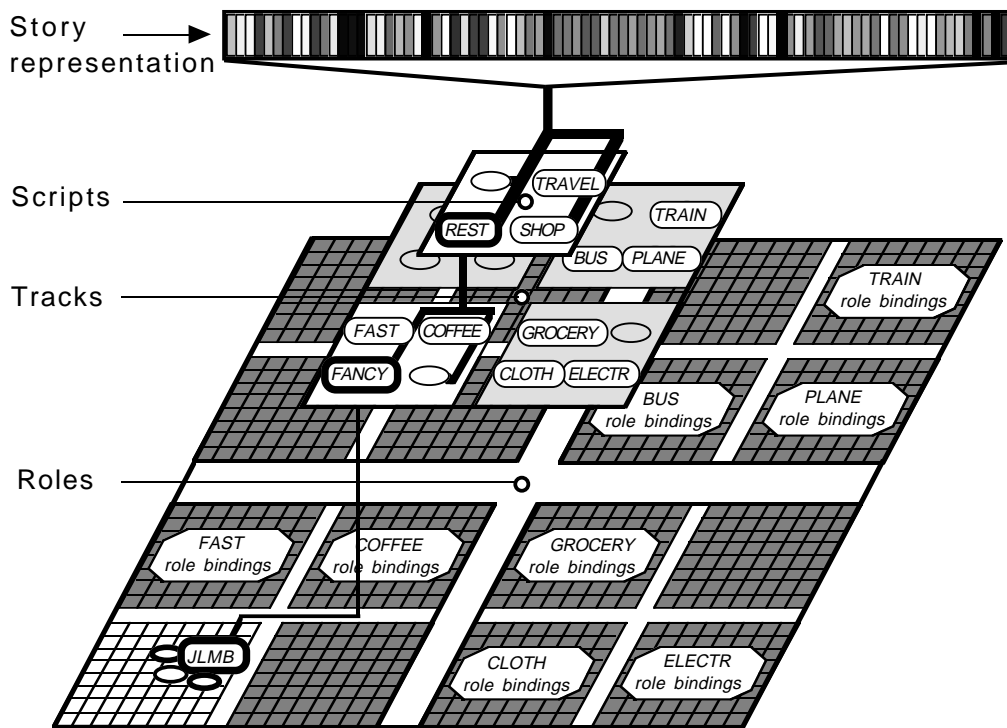


Figure 5: **The hierarchical feature map classification of script-based stories.** Labels indicate the maximally responding unit for the different scripts and tracks. This particular input story representation is classified as an instance of the restaurant script (top level) and fancy-restaurant track (middle level), with role bindings customer=John, food=lobster, restaurant=MaMaison, tip=big (i.e., unit JLMB, bottom level). Before passing on the originally 84-component representation vector to the next level, the REST-unit removes those 22 components that do not vary across the different restaurant stories, and the FANCY-unit removes those 44 components of the remaining vector that do not vary across the different fancy-restaurant stories. At the bottom level, only an 18 component vector representing the role bindings remains to be mapped. Compression is determined automatically based on the variance in the components, and varies slightly depending on the script and the track.

Hierarchical feature maps have a number of properties that make them useful for memory organization: (1) The organization reflects the properties of the data. The hierarchy corresponds to the levels of variation in the data, and the maps lay out the similarities of each level; (2) The most salient components of the input data are singled out, and more resources are allocated for representing them accurately; (3) The classification is very robust, and usually correct even if the input vector is noisy or incomplete; (4) The organization is formed in an unsupervised manner, extracting it from input examples; and (5) Self-organizing a hierarchy of small maps instead of a single large one means dividing the classification task into hierarchical subgoals, which is an efficient way to reduce complexity.

### 2.3.2 Trace feature maps

An ordinary feature map is a classifier, mapping an input vector onto a location in the map. A trace feature map, in addition, creates a memory trace at 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



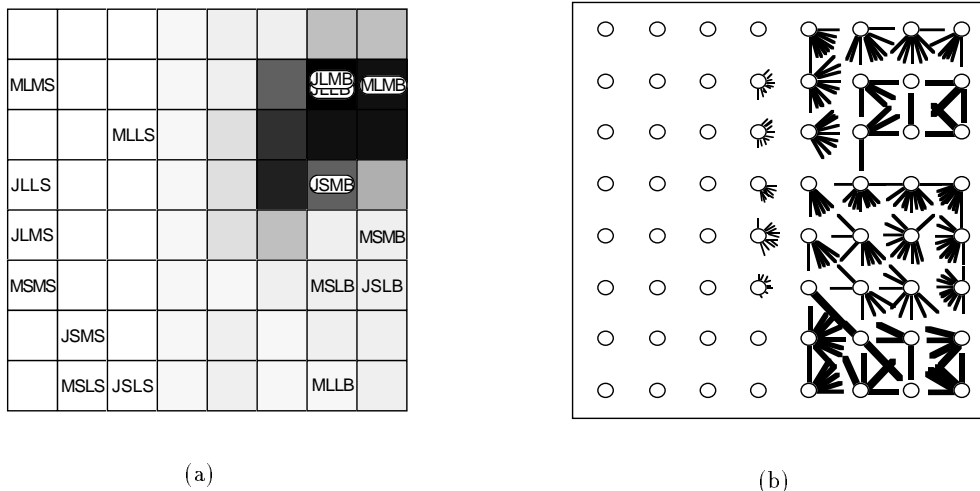


Figure 6: **The trace feature map for fancy-restaurant stories.** (a) The organization of the map. Each story in the test set were mapped on one unit in the map, labeled by the role bindings (where J=John, M=Mary, L=lobster, S=steak, M=Mamaison, L=Leone’s, B=big, and S=small (for size of tip)). The activation of the units (shown in gray-scale) indicates retrieval of the JLMB story. (b) Lateral connections after storing first JLMB and then MLLB. Line segments indicate excitatory lateral connections originating from each unit, length and width proportional to the magnitude of the weight. Inhibitory connections are not shown. The latter trace has partially obscured the earlier one.

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 6b).

A stored vector is retrieved by giving the map an approximation of the vector as a cue. The initial response is again a localized activity pattern, and because the map is topological, it is likely to be located somewhere near the stored trace. If the cue is close enough, the lateral connections pull the activity to the center of the trace (figure 6a), and the external input weights of the most highly active unit give the stored vector. If the cue is too far, the initial 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 feature map can complete a partial cue, and indicate when there is no appropriate trace in the memory.

The trace feature map exhibits interesting memory effects that result from interactions between traces. Later traces steal units from earlier ones, making later traces more likely to be retrieved (figure 6b). 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 that are unique, that is, located in a sparse area of the map, are not affected, no matter how old they are. Similar effects are common in human long-term memory [2; 39].

### 2.3.3 Storage and retrieval

A story is represented in the episodic memory by the maximally responding units at the script, track, and role-binding levels. However, each unit on a role-binding map stands for a unique story, and 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 feature 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.

## 3 Training and testing DISCERN

### 3.1 Training methodology

A good advantage of the modular architecture can be made in training the system. The tasks of the six processing modules are separable, and they can be trained separately as long as compatible I/O material (originating from the same example stories) 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. However, the tasks often have conflicting requirements. For example, it is possible that the sentence parser's task would be easier if all animate words had low values in the first component, whereas the sentence generator would rather see high values in the same component. Because all such requirements are combined in the course of training, the representations are never exactly what each individual network tries to make them. The networks have to compensate by adapting their weights. In the end, the representations and weights of all networks are in harmony: The output patterns produced by one network are exactly what the next network learned to process as its input, and the final representations reflect the combined use of the words in the six tasks.

Interestingly, although modular training is a strong AI technique, it is also a major liability for DISCERN as a neural learning model. To allow for an efficient, parallel training, the connections between modules must be cut and training must be separated from performance. There is no way to avoid this separation as long as training is based on backpropagation. DISCERN can still be seen as a plausible learning model, but only at a higher level of abstraction. The weights are learned from correlations in the examples in a process that models environmental feedback (the training input).

### 3.2 Performance results

DISCERN was trained and tested with an artificially-generated corpus of script-based stories. The data was based on of three script templates, each with three tracks: fancy, coffee-shop, and fast-food restaurant; clothing, electronics, and grocery shopping; and plane, train, and bus travel. Each story had three to five open roles: customer, restaurant, food, taste, and tip for restaurant stories, customer, shop, and item for shopping stories, and passenger, origin, destination, and distance for travel stories. The restaurant tracks contain prespecified regularities: the food is always good in a fancy restaurant, and always bad in a fast-food restaurant. In a coffee-shop restaurant, the size of the tip correlates with the food quality: good food  $\Leftrightarrow$  big tip, bad food  $\Leftrightarrow$  small tip. These regularities were set up intentionally to see if the system could use them to fill in unspecified role bindings.

A subset of words in the data were designated as prototype words (i.e. words that would later be cloned). During training, the ID part of these words were assigned randomly for each story, but consistently throughout the whole story. In effect, DISCERN learned to accept any ID pattern for these words. The test stories were then generated by specifying 2 different instances for each prototype word, which resulted in 96 different story instantiations. The entire set of stories was used to test the accuracy of the DISCERN model (tables 1 and 2). Only the three main events from each story were included to see how well DISCERN could fill in the rest of the story. In other words, the testing data differed from the training data in two ways: (1) DISCERN had (most likely) never seen the same exact ID patterns before, and (2) in testing, it only saw a few main events of each story.

Module	All words	Instances	$E_i < 0.15$	$E_{avg}$
Lexical input map	100	100	100	.0000
Semantic output map	100	100	100	.0002
Sentence parser	99	99	100	.0129
Story parser	94	91	94	.0353
Hierarchical feature maps	93	89	96	.0153
Story generator	98	92	99	.0130
Sentence generator	98	93	99	.0289
Semantic input map	95	83	98	.0065
Lexical output map	98	93	99	.0046

Table 1: **Paraphrasing performance of DISCERN.** The first data column indicates the percentage of correct words out of all output words, the second shows the percentage for words that were created from word prototypes. The third column indicates the percentage of output units within .15 of the correct value, and the last column shows the average error per output unit.

Module	All words	Instances	$E_i < 0.15$	$E_{avg}$
Lexical input map	100	100	100	.0000
Semantic output map	100	100	100	.0002
Sentence parser	99	99	99	.0168
Cue former	82	81	89	.0464
Hierarchical feature maps	93	88	96	.0160
Answer producer	97	91	98	.0167
Sentence generator	97	92	97	.0339
Semantic input map	92	79	97	.0099
Lexical output map	97	92	99	.0066

Table 2: **Question-answering performance of DISCERN.** Because some of the role bindings are unspecified in the cue, the cue former output can never be exactly correct, and the performance figures for this network are lower than for the other networks. The sentence parser and generator also show slightly less accurate performance than in paraphrasing, because the question and answer sentences are more complex and specify more word instances than the story sentences. The ratio of errors/information is about the same in the two tasks.

The complete DISCERN system performs very well in the script processing task. Missing events and role fillers are inferred whenever possible, and at the output, about 98 percent words are correct. This is rather remarkable for a chain of networks that is 9 modules long and consists of several different types of modules. None of the modules is completely accurate, and each introduces errors to the flow of information. The errors accumulate remarkably little in the chain. The modules communicate with FGREP representations that are redundant, and at each module interface, noise is efficiently filtered out, keeping the system execution on a stable path.

## 4 Examples of high-level reasoning in DISCERN

The performance tables indicate that DISCERN does well on the average, but much of the interesting reasoning it performs can only be observed from individual examples. In the following sections, several execution traces of DISCERN are presented. DISCERN reads its input stories and questions from a file and prints out log information describing the input and output of each module. The activity patterns at each I/O assembly are compared to the representations in the lexicon, and the word label of the closest match is output. DISCERN output is shown in Courier typeface. Comments printed out by DISCERN are enclosed in brackets, otherwise all words stand for activity patterns, with “\_” indicating a blank pattern. Output

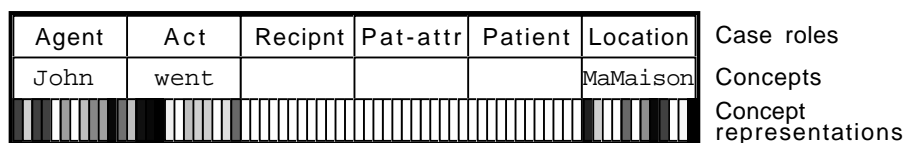


Figure 7: Case-role representation of the sentence John went to MaMaison. The word representations in each case-role correspond to those in the semantic lexicon.

layers consisting of several assemblies are enclosed between bars. The lexicon translation is indicated by “>”.

## 4.1 Parsing

Let us follow DISCERN as it processes the story about John’s visit to MaMaison:

```
[ parsing input story: ]
JOHN>John WENT>went TO>to MAMAISON>MaMaison .>.
|John went _ _ MaMaison|
JOHN>John ASKED>asked THE>the WAITER>waiter FOR>for LOBSTER>lobster .>.
|John asked waiter _ lobster _|
JOHN>John LEFT>left A>a BIG>big TIP>tip .>.
|John left waiter big tip _|

[ into internal rep: ]
|$restaurant $fancy John lobster MaMaison good big|

[ storing into episodic memory: ]
[ image units (0,0), (0,0), (7,1): ]
|$restaurant $fancy John lobster MaMaison good big|
```

This story is an instance of the fancy-restaurant track of the restaurant script. The actual input to DISCERN consists of distributed representations for the lexical word symbols. Each representation is mapped on the lexical map, and the label of the image unit is output. The first word is JOHN, which is mapped on the unit whose input weights are closest to the representation of JOHN (figure 2). The activity propagates through the associative connections to the semantic map, where the unit whose input weight vector is closest to the representation of John receives the highest activity. The activity then propagates through the input weights of this unit to the single input assembly of the sentence parser.

The same process takes place for each word of the first sentence, until a period is input. At this point, the sentence parser propagates the activity pattern at its output layer, |John went \_ \_ MaMaison| (figure 7) to the input layer of the story parser.

The same process repeats for each sentence in the story. After the last sentence, the activity pattern at the output layer of the story-parser network, |\$restaurant \$fancy John lobster MaMaison good big| (figure 8), indicates the complete internal representation of the story. This representation is given to the hierarchical feature map system (figure 5), which classifies it as an instance of the restaurant script (map (0,0)) and fancy-restaurant track (submap (0,0)), and maps it on unit (7,1) on this map, creating a trace in the process. The combined weights of the image units at the three levels of the hierarchy constitute the representation of the story in the episodic memory, |\$restaurant \$fancy John lobster MaMaison good big|.

Note that the story parser has built a correct representation for the story even though a number of events that are part of the fancy-restaurant script were not mentioned in the input. In general, it seems that a network that builds a stationary representation of a sequence might be quite sensitive to omissions and changes. Each input item is interpreted against the current position in the sequence by combining it with the previous hidden layer. If items are missing or in wrong order, it seems that the system could very easily lose track. But this is not the case in the story parser. The network was trained always to shoot for the complete output representation, which means that most of the work is done when the first sentence is read. Subsequent sentences have very little effect on the hidden layer pattern, and consequently omissions and small changes in the sequence are not critical. The strong filling-in capability (by generating expectations


Script	Track	R/Cstmr	R/Food	R/Restr	R/Taste	R/Tip	Roles
\$restr	\$fancy	John	lobster	MaMaison	good	big	Fillers
							Concept rep.

Figure 8: Representation of the story by its role bindings. The assemblies are data-specific: their interpretation depends on the pattern in the script slot. The role names R/... are specific for the restaurant script.

and defaults) is due to the fact that there is very little variation in the training sequences. But it is exactly this lack of variety that makes up scripts: they are stereotypical sequences of events. Interestingly, it follows that filling in the unmentioned events is an easy task for a sequential network system such as DISCERN.

Once the script representation has been formed and stored in the memory, there is no way of knowing which events were actually mentioned in the story. What details are produced in the paraphrase and question answering depends on the training of the output networks. This result is consistent with psychological data on how people remember stories of familiar event sequences [3; 17; 18]. The distinction of what was actually mentioned and what was inferred becomes blurred. Questions or references to events that were not mentioned are often answered as if they were part of the original story.

## 4.2 Paraphrasing

The generator subsystem reverses the parsing process. The episodic memory representation is fed to the input layer of the story-generator network, and DISCERN now generates a fully expanded paraphrase of the story:

```
[ generating paraphrase: ]
|John went _ _ _ MaMaison|
John>John>JOHN went>went>WENT to>to>TO MaMaison>MaMaison>MAMAISON .>.>.
|waiter seated _ _ John _|
the>the>THE waiter>waiter>WAITER seated>seated>SEATED John>John>JOHN .>.>.
|John asked waiter _ lobster _|
John>John>JOHN asked>asked>ASKED the>the>THE waiter>waiter>WAITER
for>for>FOR lobster>lobster>LOBSTER .>.>.
|John ate _ good lobster _|
John>John>JOHN ate>ate>ATE a>a>A good>good>GOOD lobster>lobster>LOBSTER .>.>.
|John paid waiter _ _ _|
John>John>JOHN paid>paid>PAID the>the>THE waiter>waiter>WAITER .>.>.
|John left waiter big tip _|
John>John>JOHN left>left>LEFT a>a>A big>big>BIG tip>tip>TIP .>.>.
|John left _ _ _ MaMaison|
John>John>JOHN left>left>LEFT MaMaison>MaMaison>MAMAISON .>.>.
```

The story generator produces the case-role representation of the first sentence, |John went \_ \_ \_ MaMaison| as its first output pattern. This pattern propagates to the input of the sentence generator, which outputs the semantic representations of the words one at a time. Each word is given to the semantic map of the lexicon. For example, the first word **John** is mapped on the unit whose input weights are closest to the representation of **John**. The activity propagates through the associative weights to the lexical map, where the unit **JOHN** lights up strongest. The representation stored in the input weights of this unit is output as the first word in the stream of lexical output word representations. The same process repeats for each word in the sentence and each sentence in the story. Note that in the paraphrase, DISCERN has correctly included a number of events that were not mentioned in the original story. Also, because in the fancy-restaurant script the food is always good, DISCERN has inferred that the lobster was good. In the case of a coffee-shop story, it would have made a similar inference from the fact that the tip was big.

## 4.3 Basic Question Answering

The questions for DISCERN are simple inquiries about the role bindings of the story, such as **What did John eat at MaMaison? Who bought shoes at Bullock's? Where did Mary take a plane to? How did**

Agent	Act	Recipnt	Pat-attr	Patient	Location
John	ate			what	MaMaison

Figure 9: Case-role representation of the question `What did John eat at MaMaison?` Questions are represented as sentences, but processed through a different pathway.

Script	Track	R/Cstmr	R/Food	R/Restr	R/Taste	R/Tip
\$restr	\$fancy	John	(?)	MaMaison	good	(?)

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

`John like the lobster at MaMaison?` and `Did John travel a big distance?` Of course, DISCERN can be trained with any type of question, and as long as it can be answered with the information the networks have, the system will learn to answer it. However, often even these simple questions require inferences beyond the original incomplete story, demonstrating script-based inference.

At this point, DISCERN goes on to read four more stories: two instances of electronics-shopping and one of coffee-shop restaurant and plane-travel. It is then asked a number of questions about these stories. The questions are presented in random order. DISCERN first has to determine which story the question refers to, and then generate, and output, an answer. Let us look at the processing of the first question in detail (for the sake of legibility, the semantic map and sentence the generator will be excluded from the log output):

```
[ parsing question: ]
WHAT DID JOHN EAT AT MAMAISON ?
|John ate _ _ what MaMaison|

[ into cue: ]
|$restaurant $fancy John lobster MaMaison good *from(big)*|

[ retrieving from episodic memory: ]
[ map (0,0), (0,0):  →(4,3):0.741 →(7,1):0.944 →(7,1):0.989
                    →(7,1):0.989 →(7,1):0.995 →(7,1):0.994
                    →(7,1):0.997 →(7,1):0.996 →(7,1):0.997 ]

[ image units (0,0), (0,0), (7,1): ]
|$restaurant $fancy John lobster MaMaison good big|

[ generating answer: ]
|John ate _ good lobster _|
JOHN ATE A GOOD LOBSTER .
```

The sentence parser reads the question sentence as usual, and forms a case-role representation of it, `|John ate _ _ what MaMaison|` (figure 9). This pattern is sent to the cue former, which makes a best guess about the story that this question refers to. A number of role bindings can be established, but the rest of them are unspecified, and the cue former generates a pattern that is an average of the different alternatives, weighted by how often each story occurred in the training data (figure 10). The average of `lobster` and `steak` happens to be closest to `lobster` (in Euclidean distance), and the average of `big` and `small` closest to `from`, and these words appear in the transcript above (incorrect words are enclosed between asterisks with the correct word in parenthesis). An interesting effect results from this tendency to generate averages: other things being equal, the system is more likely to retrieve a frequently occurring story than an infrequent one. The cue pattern is closest to the most common story representation, and if an infrequent story is to be recalled, more details need to be specified in the question. Similar frequency effect has been observed in human free recall [19; 23; 52].

The cue pattern is given to the episodic memory, which classifies it as an instance of the restaurant

Agent	Act	Recipt	Pat-attr	Patient	Location
John	ate		good	lobster	

Figure 11: Case-role representation of the answer John ate a good lobster.

script (0,0) and fancy-restaurant track (0,0). The initial response of the fancy-restaurant role-binding map is centered around the unit (4,3) with activity 0.741, but quickly settles to (7,1), where the trace was stored. The story representation (figure 8) is obtained by combining the weights of the three image units.

The answer producer takes the cue representation and the story representation as its input and generates the case-role representation of the answer, |John ate \_ good lobster \_| (figure 11). The sentence generator produces words of the answer sentence one at a time, and the lexicon translates each one of them into the corresponding lexical symbol.

Note that the question above refers to an event that was not mentioned in the original story. DISCERN was told that John went to MaMaison and ordered a lobster, but nothing was said about eating the lobster or about the quality of the food. This makes no difference to DISCERN. It read the story, immediately made the appropriate inferences, and can no longer remember what was mentioned and what was inferred. Eating is always part of the restaurant visit, and it knows that **lobster** was the food in this visit. It confidently declares that John ate the lobster and that it was good.

#### 4.4 Disambiguating the Question

Consider the following example:

```
[ parsing question: ]
WHERE DID MARY EAT FISH ?
|Mary ate _ _ fish where|

[ into cue: ]
|$restaurant $coffee Mary fish Denny's bad *from(small)*|

[ retrieving from episodic memory: ]
[ map (0,0), (1,1): →(4,3):0.733 →(3,6):0.965 →(3,6):0.993
                  →(3,6):0.993 →(3,6):0.996 →(3,6):0.996
                  →(3,6):0.998 →(3,6):0.997 →(3,6):0.998 ]

[ image units (0,0), (1,1), (3,6): ]
|$restaurant $coffee Mary fish Denny's bad small|

[ generating answer: ]
|Mary ate _ _ fish Denny's|
MARY ATE FISH AT DENNY'S .
```

This example illustrates how DISCERN uses the information provided by the question to come up with a guess for the target story. A question about eating must refer to a restaurant story, but which track? If the restaurant was mentioned, it would be easy to tell whether it is a fancy, coffee-shop, or fast-food restaurant. However, the food is just as good a cue. **Fish** is only eaten in coffee-shop restaurants in the training data, and DISCERN generates a cue for a coffee-shop story with **Mary** as the customer and **fish** as food. The rest of the role bindings are averages of the different alternatives. The cue pattern is easily mapped to the coffee-shop role-binding map, which completes the pattern by settling onto an existing trace.

At this point, DISCERN has read two different electronics-shopping, and they are stored on the same map. When the question refers to an electronics-shopping story, the retrieval process has to disambiguate between the alternative traces:

```
[ parsing question: ]
WHERE DID MARY BUY TV ?
|Mary bought _ _ tv where|
```

```

[ into cue: ]
|$shopping $electronics Mary tv CircuitCty - -|

[ retrieving from episodic memory: ]
[ map (1,0), (1,0): →(7,1):0.701 →(7,1):0.931 →(7,1):0.964
                    →(7,1):0.963 →(7,1):0.971 →(7,1):0.972
                    →(7,1):0.975 →(7,1):0.975 →(7,1):0.976 ]

[ image units (1,0), (1,0), (7,1): ]
|$shopping $electronics Mary tv CircuitCty - -|

[ generating answer: ]
|Mary bought - - tv CircuitCty|
MARY BOUGHT TV AT CIRCUITCTY .

[ parsing question: ]
WHERE DID JOHN BUY TV ?
|John bought - - tv where|

[ into cue: ]
|$shopping $electronics John tv RadioShack - -|

[ retrieving from episodic memory: ]
[ map (1,0), (1,0): →(1,6):0.710 →(1,6):0.937 →(1,6):0.972
                    →(1,6):0.969 →(1,6):0.975 →(1,6):0.974
                    →(1,6):0.976 →(1,6):0.975 →(1,6):0.975 ]

[ image units (1,0), (1,0), (1,6): ]
|$shopping $electronics John tv RadioShack - -|

[ generating answer: ]
John bought - - tv RadioShack
JOHN BOUGHT TV AT RADIOSHACK .

```

Both of these questions are classified as references to electronics shopping stories. The customer is specified in both questions, and the filler of this role is different in the two traces. The initial response of the electronics shopping map is closer to the trace with the matching customer, and the activity settles around this trace. In other words, the story that better matches the question is retrieved.

## 4.5 Ambiguous Questions

If the cue is so inaccurate that it is not possible to determine the correct script and the track, the retrieval process becomes slightly more complicated. DISCERN has to search for the appropriate trace on several role-binding maps. The response of each high-level unit is compared to a search threshold parameter to decide which maps are worth looking into. If the question determines a unique story in memory, only one of the submaps leads to a trace at the bottom level. The activity on the other maps does not settle, indicating “not found.” If several traces are found, the one closest to the cue will develop the highest activity, and will be selected. In the following examples, the cue is deliberately incomplete and ambiguous. The middle-level search threshold parameter was increased (from 1.0 to 3.0) so that DISCERN can look at maps that do not match the cue as well as the cues in the previous examples.

```

[ parsing question: ]
WHO ATE ?
|who ate - - *(fish)* Denny's|

[ into cue: ]
|$restaurant $coffee *John(Mary)* *spaghetti(fish)* Denny's *(bad)* *from(small)*|

[ retrieving from episodic memory: ]
[ map (0,0), (0,0): →(3,3):0.741 →(7,1):0.933 →(7,1):0.986
                    →(7,1):0.986 →(7,1):0.993 →(7,1):0.992
                    →(7,1):0.996 →(7,1):0.995 →(7,1):0.997 ]
[ map (0,0), (0,1): →(1,0):0.701 →(1,0):0.000 →(1,0):0.701
                    →(1,0):0.000 ]
[ map (0,0), (1,1): →(4,3):0.737 →(3,6):0.963 →(3,6):0.993
                    →(3,6):0.993 →(3,6):0.996 →(3,6):0.996
                    →(3,6):0.998 →(3,6):0.997 →(3,6):0.998 ]

[ image units (0,0), (1,1), (3,6): ]
|$restaurant $coffee Mary fish Denny's bad small|

[ generating answer: ]
|Mary did - - -|
MARY DID .

```



Compared to the questions DISCERN was trained on, this question is incompletely specified, with several words left out from the end. These words normally specify the track, but now this information is not available. The sentence parser and the cue former produce patterns that are averages of the possible alternatives. A question about eating refers to a restaurant story, but everything else is a guess. DISCERN looks at all role-binding maps under the restaurant script for possible traces. It finds nothing in the fast-food map (0, 1) but both the fancy and the coffee-shop maps settle to a stored trace. The response of the fancy-restaurant map is stronger in this case, and it is selected.

Either one of the restaurant stories can be reliably retrieved if more is specified in the question:

```
[ parsing question: ]
WHO ATE LOBSTER ?
|who ate _ _ lobster *Leone's(MaMaison)*|

[ into cue: ]
|$restaurant $fancy John steak *Leone's(MaMaison)* good *from(big)*|

[ retrieving from episodic memory: ]
[ map (0,0), (0,0):  →(4,3):0.739 →(7,1):0.941 →(7,1):0.988
                    →(7,1):0.988 →(7,1):0.994 →(7,1):0.994
                    →(7,1):0.996 →(7,1):0.995 →(7,1):0.997 ]
[ map (0,0), (1,1):  →(5,3):0.712 →(3,6):0.909 →(3,6):0.980
                    →(3,6):0.973 →(3,6):0.987 →(3,6):0.982
                    →(3,6):0.990 →(3,6):0.985 →(3,6):0.991 ]

[ image units (0,0), (0,0), (7,1): ]
|$restaurant $fancy John lobster MaMaison good big|

[ generating answer: ]
|John did _ _ _ _|
JOHN DID .

[ parsing question: ]
WHAT DID MARY EAT ?
|Mary ate _ _ what Denny's|

[ into cue: ]
|$restaurant $coffee Mary fish Denny's *. (bad)* *from(small)*|

[ retrieving from episodic memory: ]
[ map (0,0), (0,0):  →(6,3):0.697 →(7,1):0.920 →(7,1):0.983
                    →(7,1):0.983 →(7,1):0.992 →(7,1):0.990
                    →(7,1):0.995 →(7,1):0.993 →(7,1):0.996 ]
[ map (0,0), (0,1):  →(1,6):0.713 →(1,6):0.000 →(1,6):0.713
                    →(1,6):0.000 ]
[ map (0,0), (1,1):  →(4,3):0.734 →(3,6):0.966 →(3,6):0.994
                    →(3,6):0.993 →(3,6):0.996 →(3,6):0.996
                    →(3,6):0.998 →(3,6):0.997 →(3,6):0.998 ]

[ image units (0,0), (1,1), (3,6): ]
|$restaurant $coffee Mary fish Denny's bad small|

[ generating answer: ]
|Mary ate _ bad fish _|
MARY ATE A BAD FISH .
```

The first question reveals that food was of the fancy type. The cue former can now generate an accurate representation for the script, track, R/Food, and R/Taste. The correct trace creates an even stronger response than before, and is selected. In the second question, the customer is specified. This does not specify the track, but it is enough to disambiguate between the two traces, because they have different customers. The fancy trace with customer=John responds more weakly and the coffee-shop trace with customer=Mary responds more strongly than before, and the coffee-shop story is selected.

In the following example, there are again two traces competing for the same question, but now the traces are located on the same role-binding map:

```
[ parsing question: ]
WHO BOUGHT TV ?
|who bought _ _ tv RadioShack|

[ into cue: ]
|$shopping $electronics John tv *CircuitCty(RadioShack)* _ _|

[ retrieving from episodic memory: ]
[ map (1,0), (1,0):  →(6,4):0.703 →(1,6):0.913 →(1,6):0.959
                    →(1,6):0.953 →(1,6):0.961 →(1,6):0.959
                    →(1,6):0.962 →(1,6):0.960 →(1,6):0.961 ]
```

```

[ image units (1,0), (1,0), (1,6): ]
|$shopping $electronics John tv RadioShack _ _|

[ generating answer: ]
|John did _ _ _ _|
JOHN DID .

```

As the reader may recall, DISCERN has read two electronics shopping stories. In the first one, Mary bought a TV at CircuitCty, while in the second one John bought one at RadioShack. The question is completely ambiguous regarding the target story. However, the traces are located on the same area of the map, and the later trace has stolen units from the earlier one. It has a wider attractor basin in that general area, and an ambiguous cue is drawn to the center of the later trace. As a result, **John** is returned as the customer. The older trace still exists, but to get to it, it is necessary to include more information in the question, as was done by asking **WHERE DID MARY BUY TV** in section 4.4 above.

In general, incomplete questions can be used to cue DISCERN just as well as the complete questions it was trained on. The sentence parser and the cue former will use whatever information was specified in the question to come up with the best guess. The episodic memory then determines the story trace that best matches the cue. If the question is ambiguous between several stories in the same map, the most recent one is retrieved. Among possible stories in different maps the one that responds most strongly is selected.

Enumerating the possible traces (e.g. **JOHN DID AND MARY DID**) would also be a plausible response to the ambiguous question. Currently, there is no easy way to do that in DISCERN. Combining multiple traces into a single answer requires higher-level control, which is very difficult to implement through simple interactions between modules. A separate control network would need to be built that would monitor the retrieval process and decide how to output the result (section 6.1).

## 4.6 Misleading and Incorrect Questions

DISCERN can also answer questions that provide slightly inaccurate information. Incorrect role bindings in the question are automatically corrected during retrieval:

```

[ parsing question: ]
HOW DID JOHN LIKE STEAK AT MAMAISON ?
|John thought _ what *steak(lobster)* MaMaison|

[ into cue: ]
|$restaurant $fancy John *steak(lobster)* MaMaison good *from(big)*|

[ retrieving from episodic memory: ]
[ map (0,0), (0,0):  →(4,3):0.750 →(7,1):0.943 →(7,1):0.988
                    →(7,1):0.989 →(7,1):0.995 →(7,1):0.994
                    →(7,1):0.997 →(7,1):0.996 →(7,1):0.997 ]

[ image units (0,0), (0,0), (7,1): ]
|$restaurant $fancy John lobster MaMaison good big|

[ generating answer: ]
|John thought _ good lobster MaMaison|
JOHN THOUGHT LOBSTER WAS GOOD AT MAMAISON .

```

The question assumes that John ate a steak at MaMaison, although he actually had lobster. The cue representation accurately represents **steak**. However, the only trace available in the fancy-restaurant map has **lobster** as food. Otherwise this trace matches the cue very well, and the activity is drawn to the center of the trace. In the retrieved vector, **lobster** is accurately represented. The fancy-food in the retrieved representation has a stronger correlation with the correct answer, and the answer producer also correctly outputs **lobster**. In other words, DISCERN decided that because John's lobster meal at MaMaison was the only appropriate trace in memory, that must be the target of the question, and the question itself was probably inaccurate.

This example illustrates an important point: the questions do not provide hard constraints on retrieval in DISCERN. There is no difference between an incomplete cue and an incorrect one. If the question as a whole is close enough to a stored trace, it will be retrieved. Some information in the question is automatically completed in the process, and some other information may be overridden. DISCERN does recognize, though, when a question is too different from anything in the memory, and should not be corrected:

```

[ parsing question: ]
WHAT DID JOHN TAKE TO JFK ?
|John took _ _ what jfk|

[ into cue: ]
|$travel $plane John *dfw(lax)* jfk big _|

[ retrieving from episodic memory: ]
[ map (1,1), (1,0): →(6,5):0.728 →(1,0):0.313 →(6,5):0.709
                  →(1,0):0.412 ]

[ oops: no image found ]

```

There is one plane-travel story in the memory, about Mary's trip to SFO. A question about John's flight to JFK has nothing in common with the trace, other than the fact that they are classified on the same map. The initial activity pattern does not settle to the trace, and DISCERN refuses to answer the question.

## 4.7 Incorrect Event Order

In many stories, the order of certain events is not crucial. For example in a coffee-shop restaurant, the customer leaves the tip at the table and then pays at the cashier, but these events could take place in the opposite order as well. DISCERN is robust against such minor changes:

```

[ parsing input story: ]
JOHN WENT TO DENNY'S .
|John went _ _ _ Denny's|
JOHN ASKED THE WAITER FOR FISH .
|John asked waiter _ fish _|
JOHN ATE A BAD FISH .
|John ate _ bad fish _|
JOHN PAID THE CASHIER .
|John paid cashier _ _ _|
JOHN LEFT A SMALL TIP .
|John left waiter small tip _|

[ into internal rep: ]
|$restaurant $coffee John fish Denny's bad small|

[ storing into episodic memory: ]
[ image units (0,0), (1,1), (3,6): ]
|$restaurant $coffee John fish Denny's bad small|

[ generating paraphrase: ]
|John went _ _ _ Denny's|
JOHN WENT TO DENNY'S .
|John seated _ _ John _|
JOHN SEATED JOHN .
|John asked waiter _ fish _|
JOHN ASKED THE WAITER FOR FISH .
|John ate _ bad fish _|
JOHN ATE A BAD FISH .
|John left waiter small tip _|
JOHN LEFT A SMALL TIP .
|John paid cashier _ _ _|
JOHN PAID THE CASHIER .
|John left _ _ _ Denny's|
JOHN LEFT DENNY'S .

```

The first few events establish the right context for the story, and subsequent events have only a small effect on the hidden-layer pattern of the story parser. They may appear slightly out of order without much problem. The story parser is still able to generate the correct slot-filler representation for the story, and in the paraphrase, the events appear in normal order. This behavior matches human performance in recalling script-based stories [3].

However, the hidden layer pattern does undergo a subtle change as more events are input. If an event appears very far from its correct place, the network will have trouble interpreting it. The problem is even worse if the role bindings are established out of order:

```

[ parsing input story: ]
JOHN WENT TO DENNY'S .
|John went _ _ _ Denny's|

```

```

JOHN ASKED THE WAITER FOR FISH .
|John asked waiter _ fish _|
JOHN LEFT A SMALL TIP .
|John left waiter small tip _|
JOHN ATE A BAD FISH .
|John ate _ bad fish _|
JOHN PAID THE CASHIER .
|John paid cashier _ _ _|

[ into internal rep: ]
|$restaurant $coffee John fish *Norms(Denny's)* bad small|

[ storing into episodic memory: ]
[ image units (0,0), (1,1), (4,7): ]
|$restaurant $coffee John fish *Norms(Denny's)* bad small|

```

In this example, John leaves the tip before eating the food, and DISCERN is confused. Note that such an event order would confuse a human reader also, because it violates the causal chain of the script. The food quality must be established first before deciding how large the tip should be. In terms of the network behavior, the hidden layer pattern undergoes a significant change after learning that the tip was small, and the network has trouble interpreting the event that establishes taste=**bad** in the novel context.

## 4.8 Dealing with Incomplete Input

What happens when the input story does not provide enough information to instantiate a role binding? Consider the following example:

```

[ parsing input story: ]
MARY WENT TO BUS-STOP .
|Mary went _ _ _ bus-stop|
MARY WAITED FOR THE BUS .
|Mary waited _ _ bus _|
MARY PAID THE DRIVER .
|Mary paid driver _ _ _|
MARY GOT-OFF THE BUS .
|Mary got-off _ _ _ bus|

[ into internal rep: ]
|$travel $bus Mary bus-stop beach small _|

[ storing into episodic memory: ]
[ image units (1,1), (0,0), (5,0): ]
|$travel $bus Mary bus-stop beach small _|

```

The input story does not specify where Mary took the bus to. The story parser generates an output pattern that is an average of all alternatives in the destination slot. The content part of this pattern is an accurate representation of the content of **bus-destination**, but the IDs have average values. However, there is always some noise in the output, and the ID values are never exactly equal to the average. In this case, the pattern is slightly closer to **beach** than to **downtown**, the other alternative. When the story representation is given to the episodic memory, it is mapped on one of the existing representations on the role-binding map, and the destination pattern becomes an even better representation of **beach**.

DISCERN has now made the guess that the destination was **beach**, and no longer considers other alternatives. In the paraphrase and in question answering it consistently maintains this assumption:

```

[ parsing question: ]
WHERE DID MARY TAKE A BUS TO ?
|Mary took _ _ bus where|

[ into cue: ]
|$travel $bus Mary *bus-termin(bus-stop)* beach small _|

[ retrieving from episodic memory: ]
[ map (1,1), (0,0):
  →(7,1):0.669 →(5,0):0.899 →(5,0):0.979
  →(5,0):0.975 →(5,0):0.988 →(5,0):0.983
  →(5,0):0.991 →(5,0):0.986 →(5,0):0.992 ]

[ image units (1,1), (0,0), (5,0): ]
|$travel $bus Mary bus-stop beach small _|

```

```
[ generating answer: ]
|Mary took _ _ bus beach|
MARY TOOK A BUS TO BEACH .
```

Unfortunately, DISCERN maintains the assumption even when the question supplies the missing information:

```
[ parsing question: ]
DID MARY TRAVEL A BIG DISTANCE TO DOWNTOWN ?
|Mary traveled _ big distance *downtown(beach)*|

[ into cue: ]
|$travel $bus Mary *bus-termin(bus-stop)* *downtown(beach)* small _|

[ retrieving from episodic memory: ]
[ map (1,1), (0,0): →(7,1):0.689 →(5,0):0.868 →(5,0):0.971
                    →(5,0):0.967 →(5,0):0.984 →(5,0):0.977
                    →(5,0):0.997 →(5,0):0.981 →(5,0):0.990 ]

[ image units (1,1), (0,0), (5,0): ]
|$travel $bus Mary bus-stop beach small _|

[ generating answer: ]
|Mary traveled _ small distance _|
MARY TRAVELED A SMALL DISTANCE .
```

The question assumes that the destination was **downtown**. As the retrieved representation indicates, DISCERN still maintains its initial guess (destination=**beach**, although this does not show up in the actual answer). DISCERN treats the new information provided by the question simply as a mistake, and corrects it. Of course, DISCERN could be right. However, in situations like this it would be more plausible to incorporate the new information into the memory trace [11; 29]. If the question assumes a fact that was originally left unspecified, it is plausible to incorporate the fact into the memory. Currently DISCERN has no mechanism for doing this. After the trace has been created, it cannot be modified. All discrepancies between the question and the trace are treated as errors in the question.

Similar unyielding but consistent behavior occurs when DISCERN generates an incorrect role binding due to an internal processing error. For example, it could have generated destination=**beach** even if **downtown** was specified in the input story. Most of the (few) errors that DISCERN makes are of this type. The reason is that whereas semantic errors (i.e. confusions between word types) are immediately cleaned up by later modules, confusions between word instances cannot be similarly corrected. **Beach** is a perfectly plausible role binding for destination, consistent with everything else in the representation, and the later modules accept it as it is. Interestingly, because all sentences are generated from the same story representation pattern, even if a role binding is incorrect, it is usually consistent throughout the paraphrase and question answering. Thus, DISCERN *performs plausible role bindings*: an essential task in high-level inferencing and postulated as very difficult for parallel distributed processing systems to achieve [12].

## 4.9 Forgetting and Memory Confusions

Memory errors occur when the episodic memory becomes overloaded with similar stories. Let us have DISCERN read two almost identical narratives:

```
[ parsing input story: ]
MARY WENT TO BULLOCK'S .
|Mary went _ _ _ Bullock's|
MARY LOOKED FOR GOOD JEANS .
|Mary looked _ good jeans _|
MARY TOOK THE BEST JEANS .
|Mary took _ best jeans _|

[ into internal rep: ]
|$shopping $clothing Mary jeans Bullock's _ _|

[ storing into episodic memory: ]
[ image units (1,0), (0,0), (1,0): ]
|$shopping $clothing Mary jeans Bullock's _ _|

[ parsing input story: ]
MARY WENT TO BROADWAY .
|Mary went _ _ _ Broadway|
```

```

MARY LOOKED FOR GOOD JEANS .
|Mary looked _ good jeans _|
MARY TOOK THE BEST JEANS .
|Mary took _ best jeans _|

[ into internal rep: ]
|$shopping $clothing Mary jeans Broadway _ _|

[ storing into episodic memory: ]
[ image units (1,0), (0,0), (3,0): ]
|$shopping $clothing Mary jeans Broadway _ _|

```

In the first story, Mary bought jeans from Bullock's, and in the later story from Broadway. These stories are mapped very close on the clothing shopping map, and the later trace almost completely wipes out the earlier trace. DISCERN does not remember Mary's visit to Bullock's anymore:

```

[ parsing question: ]
WHAT DID MARY BUY AT BULLOCK'S ?
|Mary bought _ _ what Bullock's|

[ into cue: ]
|$shopping $clothing Mary *shoes(jeans)* Bullock's _ _|

[ retrieving from episodic memory: ]
[ map (1,0), (0,0): →(3,7):0.676 →(1,0):0.858 →(3,0):0.957
→(3,0):0.942 →(3,0):0.973 →(3,0):0.956
→(3,0):0.977 →(3,0):0.962 →(3,0):0.979 ]

[ image units (1,0), (0,0), (3,0): ]
|$shopping $clothing Mary jeans *Broadway(Bullock's)* _ _|

[ generating answer: ]
|Mary bought _ _ jeans _|
MARY BOUGHT JEANS .

```

When asked about it, DISCERN assumes that Bullock's was mentioned in the question by accident, and answers the question with clothing-store=**Broadway**. With the stories sufficiently similar, the memory has become overloaded with just two stories.

Another interesting type of memory confusion may occur when several similar narratives are stored on the same area of a role-binding map, but far enough apart so that the earlier traces are not completely erased. There is a lot of excitatory lateral support on that area of the map, and the retrieval activity may not converge to any one of the stored traces. Instead, the activity peak may develop around some intermediate unit. In this case, a representation that is a combination of the stored traces is retrieved. In other words, DISCERN remembers several distinct features of similar stories, but cannot keep the stories separate. Similar behavior has been observed experimentally in human memory [3; 56].

## 5 Discussion

DISCERN serves to illustrate and focus several important issues in subsymbolic natural language processing, including the nature of schema representations and intuitive inference, the need for type/token information, and problems in generalizing into novel situations. These issues are discussed below, after a brief overview of related work.

### 5.1 Related work on subsymbolic script processing

Very few subsymbolic models of natural language understanding have been built so far. Isolated, low-level language tasks have often been used as a domain for demonstrating new neural network techniques, but contributing to the understanding of the task has played a lesser role. There are a few models that address issues in script-based understanding. They typically take some part of the task and apply subsymbolic mechanisms to it. For example, mechanisms for processing causal sequences [16], role binding [10; 9], sequential activation and deactivation of multiple scripts [48], and learning the script taxonomy from examples [32] have been developed. Harris and Elman [21] trained a simple recurrent network to predict the next word

in a script-based story, and showed that role bindings take place not between two different occurrences of a variable, but between the variable and a global state. St. John's [50; 51] story gestalt model demonstrated how script-based regularities in the input data implement soft constraints that are brought together to form an interpretation of the text.

Such models illustrate various aspects of learning, representing, and applying schematic knowledge, but they are not self-contained story-processing systems. In contrast, DISCERN aims at integrated natural language processing in the same sense as the symbolic artificial intelligence models. This requires dividing the task into structured subtasks, building and processing internal representations, and parsing and generating natural language. The goal is to demonstrate understanding of multisentential connected text. In this sense, DISCERN aims at natural language processing at a different level than previous subsymbolic models.

## 5.2 Symbolic and subsymbolic schemas

There is an important distinction between scripts (or more generally, schemas) in symbolic systems [7; 11; 46], and scripts in subsymbolic systems such as DISCERN and the above models. In the symbolic approach, a script is stored in memory as a separate, exact knowledge structure, coded by the knowledge engineer. The script has to be instantiated by searching the schema memory sequentially for a structure that matches the input. After instantiation, the script is active in the memory and later inputs are interpreted primarily in terms of this script. Deviations are easy to recognize and can be taken care of with special mechanisms.

In the subsymbolic approach, schemas are based on statistical properties of the training examples, extracted automatically during training. The resulting knowledge structures do not have explicit representations. For example, a script exists in a neural network only as statistical correlations coded in the weights. Every input is automatically matched to every correlation *in parallel*. There is no all-or-none instantiation of a particular knowledge structure. The strongest, most probable correlations will dominate, depending on how well they match the input, but all of them are simultaneously active at all times (see also [45]). The correlations vary in scope from cooccurring case-role fillers to long sequences of events, even to regularities at the goal/plan level [10; 28; 47]. Regularities that make up scripts can be particularly well captured by such correlations, making script-based inference a good domain for the subsymbolic approach. Generalization and graceful degradation give rise to inferencing that is intuitive, immediate, and occurs without conscious control, similar to script-based inference in humans. On the other hand, it is very difficult to recognize deviations from the script and initiate exception processing when the automatic mechanisms fail. Such sequential reasoning would require intervention of a high-level "conscious" monitor (section 6.1).

The hierarchical feature map system can be seen as a general model of schema-based episodic memory, similar in spirit to the symbolic CYRUS and IPP models [26; 27]. In all these models, memory is organized according to similarities in the stories, and these similarities are more abstract than simple fillers. The memory builds schemas (by abstracting individual stories), and organization proceeds by specializing existing schemas. Much of the information is stored at the higher levels, and only the individual differences (role bindings) are stored at the lowest level. Retrieval requires traversing the hierarchy and reconstructing the complete representation from information at multiple levels.

The main difference between DISCERN and the symbolic episodic memory models is that the organization in DISCERN is based on statistical similarities in the data, whereas the symbolic models are organized according to individual examples. IPP, CYRUS, and similar models begin with predetermined initial structure, and the memory structure increases indefinitely as more experience comes in. In contrast, hierarchical feature maps are organized with a large number of examples before any traces are created. The whole organization is extracted from examples; it is not necessary to supply basic templates to bootstrap the system. The result is a stable layout of the whole space of possible experiences. However, if an input experience does not fit the existing structure, it cannot be stored correctly. How to encode novel episodes in such a statistical memory organization is currently an open problem.

### 5.3 Subsymbolic Inference

It seems that people have two fundamentally different mechanisms at their disposal for inferencing. The relevant data can be searched for using a sequential symbolic strategy. One does not have an immediate answer to the problem, but the answer is sequentially constructed from stored knowledge by a high-level goal-directed process, that is, by reasoning. Another type of inference occurs through associations immediately, in parallel, and without conscious control, in other words, 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 [22; 57]). The network extracts statistical knowledge from the input data, and these statistics are brought together to generate the inference. For example, the quality of food in the restaurant story is inferred from the whole story, not just by looking at some part and applying a specific rule. If the tip was small, the network usually infers that the food was bad, but if the customer ate a lobster, the network immediately assumes it was good, because lobsters are usually eaten in fancy restaurants that serve good food. In other words, neural networks perform probabilistic inference by producing an answer that is the average of all alternatives, weighted by their probabilities. This is exactly what is needed in, for example, filling in the missing events and fillers in a script-based story. On the other hand, intuitive inference is not useful when trying to understand unusual events, or to perform logical reasoning.

### 5.4 Processing Types and Tokens

The ID+content mechanism in DISCERN is a first step toward grounding logical (symbolic) reasoning in subsymbolic systems. Any symbolic system needs to be able to deal with two kinds of information: (1) semantics of symbols, defined as knowledge about the properties of symbols and the relationships between them; and (2) identities of symbols, based on some unique surface-level tag such as a sensory perception of the referent (see also [20]). 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  $C$  and properties  $P_1, P_2$ , and  $P_3$ , if  $x \in C$ ,  $P_1(x) \wedge P_2(x) \Rightarrow P_3(x)$ . Furthermore, it knows the facts  $P_1(x_1)$  for  $x_1 \in C$  and  $P_2(x_2)$  for  $x_2 \in C$ . In order to decide whether  $P_3$  holds, the system must be able to verify whether  $x_1$  and  $x_2$  actually refer to the same object. Establishing that the representations of  $x_1$  and  $x_2$  have similar statistical properties (i.e. content) does not allow drawing the conclusion, even if the knowledge of  $P_1(x_1)$  and  $P_2(x_2)$  is based on these same statistical properties.

A common approach is to explicitly separate the two kinds of information. The semantic knowledge is maintained in “types,” and each symbol is created as a “token,” an instance of some type. The types form semantic hierarchies, the instances inherit the properties of parent types and also accumulate specific properties of their own during processing. Whether implemented in a symbolic semantic network [40] or in a semantic network/subsymbolic neural network hybrid [53; 54; 55], in effect, there are two separate systems with a very complex interaction. One system performs reasoning based on semantic relations and the other one propagates the bindings. In the ID+content approach, on the other hand, the identity and the semantic content are kept together in a *single representation* that propagates through various pattern transformation networks. This is essential in building modular systems, because it allows modules to be connected with simple pathways. All necessary information is included in the representation, and it can be directly used as input to another module. Part of the representation is treated as the logical ID, and the rest is interpreted as the semantic content. The system is trained to process both parts simultaneously.

### 5.5 Generalization to novel inputs

Subsymbolic systems generalize fairly well to new situations that are in line with their training. For example, having seen **The man ate the lobster**, the network might be able to process **The dog ate the lobster**, if both **man** and **dog** share animate qualities in the data. However, the system would not do well with a



truly novel role binding such as **The train ate the lobster**, which goes against all regularities in the training data. The problem is that the network has no representation for an all-or-none role binding. What appears as a binding emerges from the statistical correlations between all constituents in parallel, encoded in the connections between layers. The network automatically interpolates between examples, and therefore generalizes well into familiar situations. The system can naturally model semantic illusions [14], where the actual content of the text is overridden by semantically more likely content. But truly novel role bindings would require extrapolation, which does not automatically result from correlations. Extrapolation is only possible by applying a symbolic-like higher-level rule, which is beyond the scope of current subsymbolic models.

DISCERN was trained to answer simple questions about role bindings in the stories. An interesting question is, can it generalize question types across stories? In a separate experiment, where DISCERN was trained to answer two out of the three wh-questions for each script, generalization did occur to some extent. However, there are obvious limits to the generalization that is possible. Much of it depends on the structure of the internal representation. To generalize a who-question to another representation, the same role at least needs to be represented at the same assembly. For the generalization to be meaningful, the two contexts need to have similar structure, and their representations need to be similar, even if the role names are somewhat different. For example, the customer in a restaurant corresponds to the passenger in the train.

In general, subsymbolic systems are quite limited in generalization into novel combinations of familiar input elements, such as a familiar question in a new context. Unless trained with excessive examples, the network has no reason to develop an abstract representation for the question. It only learns to process questions according to statistical correlations in the input. If DISCERN's answer-producer module is trained with who-questions about restaurant stories only, and then asked **Who bought a CD-player at Radio-Shack** in the novel context of a shopping story, the network would produce simply garbage. For generalization at this level, an abstract high-level representation for the question is necessary, and at this point it is unclear how this could be learned from examples. Some possibilities are discussed below in section 6.2.

## 6 Future work

A major contribution of DISCERN is to show that building complete natural language processing systems from subsymbolic components is feasible. DISCERN can provide a framework for future research in large modular neural network systems, and for research in connectionist language processing, episodic memory, lexicon, and semantic representation. Many issues related to these tasks have been finessed in DISCERN, and they call for further research (see [34] for a detailed discussion). For example, methods could be developed for processing more complex sentences. It might be possible to devise mechanisms and representations for sequential activation and deactivation of scripts, or even multiple simultaneously active scripts. With more advanced memory mechanisms, proactive and retroactive interference and negative transfer effects could be modeled.

Although processing simple script instantiations is a start, there is a long way to go before integrated connectionist models will rival the best symbolic AI systems. For example in story understanding, symbolic systems have been developed that analyze realistic stories in-depth, based on higher-level knowledge structures such as goals, plans, themes, affects, beliefs, argument structures, plots, and morals [1; 11; 41; 46]. Subsymbolic systems cannot yet model cognitive processes at this level. As we have seen above, they are very good at dealing with regularities and combining large amounts of simple pieces of evidence, but they do not easily lend themselves to processing complex knowledge structures and unusual and novel situations. In designing subsymbolic models that would do that, we are faced with two major problems: (1) how to implement control of complex processing strategies, and (2) how to represent and learn abstractions.

### 6.1 Connectionist control of cognitive processes

Integrated connectionist models such as DISCERN demonstrate how far it is possible to go in modeling high-level behavior by combining simple low-level processes. DISCERN is not "conscious" of what it is doing,

that is, it does not have representations concerning the nature of its own representations and processes. As a result, it cannot employ high-level strategies in controlling its processes; its behavior is limited to a series of reflex responses. With a high-level monitor and control mechanism, it would be possible to build much more powerful subsymbolic models:

1. Current systems try to process every input in exactly the same way, regardless of whether the input makes sense or not. A high-level control system could monitor the feasibility of the task and the quality of output, and initiate exception processing when the usual mechanisms fail. For example, unusual events and deviations from a script could be detected and then processed by special mechanisms.
2. Retrieval from episodic memory could be controlled intelligently. The monitor could decide how to search for traces and how to respond when multiple traces are found, or when nothing is found.
3. The monitor could keep the system on a stable path by detecting and correcting internal inaccuracies that cannot be caught by the automatic low-level filters (such as confusions between IDs in DISCERN).
4. Sequential high-level procedures and reasoning mechanisms could be implemented, such as comparing several items in the memory and applying high-level rules to conclude new information.

It might be possible to implement the necessary monitoring, modulation, and decision-making tasks with trainable control networks. These modules would receive input from several pathways in the system, monitoring its state, and their output would gate the system pathways through multiplicative connections (such as those of [37; 42]). Equipped with such mechanisms, subsymbolic models would no longer be limited to automatic reflex behavior, and would be able to perform much more robustly in the real world.

## 6.2 Representing and learning abstractions

Representing structure, processing exceptions, and making dynamic inferences are often cited as the three main challenges for subsymbolic artificial intelligence. It seems that these problems are closely related, and stem from the fundamental limitation of the current architectures in representing and learning abstractions of data.

Let us briefly review each problem. First, assembly-based representations, as they are commonly used in subsymbolic systems, must be designed in advance and they remain fixed. Units cannot be created or moved around in the network, they can only function in the same exact configuration they were trained in. It is very difficult to represent multiple fillers (e.g. **the man and the boy**), additional constituents, and recursive structures (see [10; 22; 38; 49] for possible approaches).

Second, processing knowledge in subsymbolic models is based on statistical regularities, and they cannot handle deviations from the regularities very well [35; 51]. Exceptions are simply overridden. The network has no representation for all-or-none role bindings, and as a result it cannot process truly novel inputs according to a symbolic-like higher-level rule.

The third problem, dynamic inferencing [57], is evident in trying to process questions in a new context. If the cue former and answer producer have seen, for example, a **who** question only in the context of restaurant stories, they would not be able to make sense out of it in a shopping context. The networks cannot make inferences by dynamically combining processing knowledge they have previously seen only in separate situations.

All these three problems originate from using distributed neural networks as statistical pattern transformers. The networks are trained to compute smooth functions between patterns, and as a result they can only interpolate between patterns they were trained on. They cannot represent various alternatives distinctly, and they cannot extrapolate to new patterns. In order to process novel constituents and novel structures, the networks must have meta-level information that describes the structure of the data, such as schemas, rules, and abstractions. A novel input needs to be recognized in terms of meta-level categories, and processed according to abstract knowledge in that category. In terms of the above problems:

1. A flexible representation must contain information about what the components are and how they are related.
2. Processing exceptions requires that the general rules and variable bindings be explicitly represented, so that the network can choose to use them to override the statistical regularities.
3. Dynamic inferencing is only possible if the meta-structure of the input is explicitly represented. For example, if the question-processing subsystem could represent the general structure of a **who**-question, it could apply that structure to novel contexts constructs.

Abstractions are regularities that best describe the structure of the data. It might be possible to devise a self-organizing process that is sensitive to the internal structure of the training examples. The network would learn the processing task, and at the same time develop a layout of rules, schemas, and other abstract structures that best describe the data. Further input would then be interpreted and represented in terms of this layout (i.e. in terms of the internal structure of the input). Such a capacity would be a major step toward extending subsymbolic AI beyond the limitations of current models.

## 7 Conclusion

The main conclusion from DISCERN is that subsymbolic high-level reasoning is feasible. DISCERN grounds several such phenomena in subsymbolic (statistical) processes. Learning word meanings, script processing, and episodic memory organization are based on self-organization and gradient-descent in error. Script-based inferences, expectations, and defaults automatically result from generalization and graceful degradation in distributed networks. Several types of performance errors in role binding, episodic memory, and lexical access are explained by the physical organization of the system.

DISCERN constitutes a first implementation of the integrated connectionist approach, demonstrating that it is possible to build complete systems from independently designed connectionists components. The scale-up properties of the approach seem quite good for several reasons: Hierarchical modular structure with sequential communication efficiently reduces the complexity of the high-level task. The modules filter out each other's errors, and the system performance is stable. With meaningful intermediate representations more modules can be added to the existing system. The system develops its own representations for intercommunication between modules, and these representations are optimized for the overall task. Perhaps most significantly, DISCERN processes both statistical (general, semantic) and declarative (specific, episodic) information and is able to produce sequential, symbolic high-level behavior.

The two main liabilities of DISCERN are that processing is based on a series of reflex responses, and much of the internal knowledge structures are fixed and specified in advance. Developing methods for connectionist control of high-level behavior, and for learning more of the necessary knowledge structures automatically are the two main areas for future research. Progress in these areas should eventually lead to substantially more powerful subsymbolic AI systems.

## References

- [1] S. Alvarado, M. G. Dyer, and M. Flowers. Argument representation for editorial text. *Knowledge-Based Systems*, 3:87–107, 1990.
- [2] A. D. Baddeley. *The Psychology of Memory*. Basic Books: New York, 1976.
- [3] G. H. Bower, J. B. Black, and T. J. Turner. Scripts in memory for text. *Cognitive Psychology*, 11:177–220, 1979.
- [4] A. Caramazza. Some aspects of language processing revealed through the analysis of acquired aphasia: The lexical system. *Annual Review of Neuroscience*, 11:395–421, 1988.
- [5] M. Coltheart, K. Patterson, and J. C. Marshall, editors. *Deep Dyslexia*. Routledge and Kegan Paul: London, Boston, Henley, second edition, 1988.

- [6] W. A. Cook. *Case Grammar Theory*. Georgetown University Press: Washington, DC, 1989.
- [7] R. E. Cullingford. *Script Application: Computer Understanding of Newspaper Stories*. PhD thesis, Department of Computer Science, Yale University, New Haven, CT, 1978. Technical Report 116.
- [8] G. F. DeJong. *Skimming Stories in Real Time: An Experiment in Integrated Understanding*. PhD thesis, Department of Computer Science, Yale University, New Haven, CT, 1979. Technical Report 158.
- [9] C. P. Dolan and M. G. Dyer. Symbolic schemata, role binding, and the evolution of structure in connectionist memories. In *Proceedings of the IEEE First International Conference on Neural Networks* (San Diego, CA), volume II, pp. 287–298, IEEE: Piscataway, NJ, 1987.
- [10] C. P. Dolan. *Tensor Manipulation Networks: Connectionist and Symbolic Approaches to Comprehension, Learning and Planning*. PhD thesis, Computer Science Department, University of California, Los Angeles, 1989. Technical Report UCLA-AI-89-06.
- [11] M. G. Dyer. *In-Depth Understanding: A Computer Model of Integrated Processing for Narrative Comprehension*. MIT Press: Cambridge, MA, 1983.
- [12] M. G. Dyer. Symbolic neuroengineering for natural language processing: A multilevel research approach. In J. A. Barnden and J. B. Pollack, editors, *High-Level Connectionist Models*, volume 1 of *Advances in Connectionist and Neural Computation Theory*, J. A. Barnden, series editor, pp. 32–86. Ablex: Norwood, NJ, 1991.
- [13] J. L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.
- [14] T. D. Erickson and M. E. Mattson. From words to meaning: A semantic illusion. *Journal of Verbal Learning and Verbal Behavior*, 20:540–551, 1981.
- [15] C. J. Fillmore. The case for case. In E. Bach and R. T. Harms, editors, *Universals in Linguistic Theory*, pp. 0–88. Holt, Rinehart and Winston: New York, 1968.
- [16] R. M. Golden. Representing causal schemata in connectionist systems. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pp. 13–21. Erlbaum: Hillsdale, NJ, 1986.
- [17] A. C. Graesser, S. E. Gordon, and J. D. Sawyer. Recognition memory for typical and atypical actions in scripted activities: Tests for the script pointer+tag hypothesis. *Journal of Verbal Learning and Verbal Behavior*, 18:319–332, 1979.
- [18] A. C. Graesser, S. B. Woll, D. J. Kowalski, and D. A. Smith. Memory for typical and atypical actions in scripted activities. *Journal of Experimental Psychology: Human Learning and Memory*, 6:503–515, 1980.
- [19] J. F. Hall. Learning as a function of word-frequency. *American Journal of Psychology*, 67:138–140, 1954.
- [20] S. Harnad. The symbol grounding problem. *Physica D*, 42:335–346, 1990.
- [21] C. L. Harris and J. L. Elman. Representing variable information with simple recurrent networks. In *Proceedings of the 11th Annual Conference of the Cognitive Science Society*, pp. 435–642. Erlbaum: Hillsdale, NJ, 1989.
- [22] G. E. Hinton. Mapping part-whole hierarchies into connectionist networks. *Artificial Intelligence*, 46:47–75, 1990.
- [23] W. Kintsch. *Memory and Cognition*. Wiley, New York, second edition, 1977.
- [24] T. Kohonen. *Self-Organization and Associative Memory*. Springer: Berlin, Heidelberg, New York, third edition, 1989.
- [25] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78:1464–1480, 1990.
- [26] J. L. Kolodner. *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model*. Erlbaum: Hillsdale, NJ, 1984.
- [27] M. Lebowitz. *Generalization and Memory in an Integrated Understanding System*. PhD thesis, Department of Computer Science, Yale University, New Haven, CT, 1980. Research Report 186.

- [28] G. Lee. *Distributed Semantic Representations for Goal/Plan Analysis of Narratives in a Connectionist Architecture*. PhD thesis, Computer Science Department, University of California, Los Angeles, 1991. Technical Report UCLA-AI-91-03.
- [29] W. G. Lehnert. *The Process of Question Answering*. Erlbaum: Hillsdale, NJ, 1978.
- [30] R. A. McCarthy and E. K. Warrington. *Cognitive Neuropsychology: A Clinical Introduction*. Academic Press: New York, 1990.
- [31] R. Miikkulainen. A distributed feature map model of the lexicon. In *Proceedings of the 12th Annual Conference of the Cognitive Science Society*, pp. 447–454. Erlbaum: Hillsdale, NJ, 1990.
- [32] R. Miikkulainen. Script recognition with hierarchical feature maps. *Connection Science*, 2:83–101, 1990.
- [33] R. Miikkulainen. Trace feature map: A model of episodic associative memory. *Biological Cybernetics*, 66:273–282, 1992.
- [34] R. Miikkulainen. *Subsymbolic Natural Language Processing: An Integrated Model of Scripts, Lexicon, and Memory*. MIT Press: Cambridge, MA, 1993.
- [35] R. Miikkulainen and M. G. Dyer. A modular neural network architecture for sequential paraphrasing of script-based stories. In *Proceedings of the International Joint Conference on Neural Networks* (Washington, DC), volume II, pp. 49–56. IEEE: Piscataway, NJ, 1989.
- [36] R. Miikkulainen and M. G. Dyer. Natural language processing with modular neural networks and distributed lexicon. *Cognitive Science*, 15:343–399, 1991.
- [37] J. B. Pollack. Cascaded back-propagation on dynamic connectionist networks. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, pp. 391–404. Erlbaum: Hillsdale, NJ, 1987.
- [38] J. B. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46:77–105, 1990.
- [39] L. Postman. Transfer, interference and forgetting. In J. W. Kling and L. A. Riggs, editors, *Woodworth and Schlosberg's Experimental Psychology*, pp. 1019–1132. Holt, Rinehart and Winston: New York, third edition, 1971.
- [40] M. R. Quillian. Word concepts: A theory and simulation of some basic semantic capabilities. *Behavioral Science*, 12:410–430, 1967.
- [41] J. F. Reeves. *Computational Morality: A Process Model of Belief Conflict and Resolution for Story Understanding*. PhD thesis, Computer Science Department, University of California, Los Angeles, 1991. Technical Report UCLA-AI-91-05.
- [42] D. E. Rumelhart, G. E. Hinton, and J. L. McClelland. A general framework for parallel distributed processing. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pp. 45–76. MIT Press: Cambridge, MA, 1986.
- [43] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pp. 318–362. MIT Press: Cambridge, MA, 1986.
- [44] D. E. Rumelhart and D. A. Norman. Accretion, tuning and restructuring. In J. W. Cotton and R. L. Klatzky, editors, *Semantic Factors in Cognition*, pp. 37–53. Erlbaum: Hillsdale, NJ, 1978.
- [45] D. E. Rumelhart, P. Smolensky, J. L. McClelland, and G. E. Hinton. Schemata and sequential thought processes in PDP models. In J. L. McClelland and D. E. Rumelhart, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 2: Psychological and Biological Models*, pp. 7–57. MIT Press: Cambridge, MA, 1986.
- [46] R. C. Schank and R. P. Abelson. *Scripts, Plans, Goals, and Understanding: An Inquiry into Human Knowledge Structures*. Erlbaum: Hillsdale, NJ, 1977.
- [47] N. E. Sharkey. A PDP system for goal-plan decisions. In R. Trappl, editor, *Cybernetics and Systems '88: Proceedings of the Ninth European Meeting on Cybernetics and Systems Research*, pp. 1031–1038. Kluwer: Dordrecht, Boston, 1988.

- [48] N. E. Sharkey. A PDP learning approach to natural language understanding. In I. Alexander, editor, *Neural Computing Architectures: The Design of Brainlike Machines*, pp. 92–116. MIT Press: Cambridge, MA, 1989.
- [49] P. Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence*, 46:159–216, 1990.
- [50] M. F. St. John. *The Story Gestalt: Text Comprehension by Cue-Based Constraint Satisfaction*. PhD thesis, Department of Psychology, Carnegie Mellon University, Pittsburgh, PA, 1990.
- [51] M. F. St. John. The story gestalt: A model of knowledge-intensive processes in text comprehension. *Cognitive Science*, 16:271–306, 1992.
- [52] W. H. Sumby. Word frequency and serial position effects. *Journal of Verbal Learning and Verbal Behavior*, 1:443–450, 1963.
- [53] R. A. Sumida. Dynamic inferencing in parallel distributed semantic networks. In *Proceedings of the 13th Annual Conference of the Cognitive Science Society*, pp. 913–917. Erlbaum: Hillsdale, NJ, 1991.
- [54] R. A. Sumida and M. G. Dyer. Storing and generalizing multiple instances while maintaining knowledge-level parallelism. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pp. 1426–1431. Morgan Kaufmann: San Mateo, CA, 1989.
- [55] R. A. Sumida and M. G. Dyer. Propagation filters in PDS networks for sequencing and ambiguity resolution. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4*, pp. 233–240. Morgan Kaufmann: San Mateo, CA, 1992.
- [56] P. W. Thorndyke and B. Hayes-Roth. The use of schemata in the acquisition and transfer of knowledge. *Cognitive Psychology*, 11:82–106, 1979.
- [57] D. S. Touretzky. Connectionism and compositional semantics. In J. A. Barnden and J. B. Pollack, editors, *High-Level Connectionist Models*, volume 1 of *Advances in Connectionist and Neural Computation Theory*, Barnden, J. A., series editor, pp. 17–31. Ablex: Norwood, NJ, 1991.