SCRIPT RECOGNITION WITH HIERARCHICAL FEATURE MAPS *†

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

Abstract

The hierarchical feature map system recognizes an input story as an instance of a particular script by classifying it at three levels: scripts, tracks and role bindings. The recognition taxonomy, i.e. the breakdown of each script into the tracks and roles, is extracted automatically and independently for each script from examples of script instantiations in an unsupervised self-organizing process. The process resembles human learning in that the differentiation of the most frequently encountered scripts become gradually the most detailed. The resulting structure is a hierachical pyramid of feature maps. The hierarchy visualizes the taxonomy and the maps lay out the topology of each level. The number of input lines and the self-organization time are considerably reduced compared to the ordinary single-level feature mapping. The system can recognize incomplete stories and recover the missing events. The taxonomy also serves as memory organization for script-based episodic memory. The maps assign a unique memory location for each script instantiation. The most salient parts of the input data are separated and most resources are concentrated on representing them accurately.

1 Introduction

According to script theory, people organize knowledge of everyday routines in the form of scripts (Schank and Abelson, 1977). Scripts are schemata of often-encountered, stereotypical sequences of events. Common knowledge of this kind makes it possible to efficiently perform social tasks such as visiting a restaurant, visiting a doctor, shopping at a supermarket, traveling by airplane, attending a meeting, etc. People have hundreds, perhaps thousands of scripts at their disposal. Each script divides further into different variations, or tracks. For example, there is a fancy-restaurant track, fast-food track and a coffee-shop track for the restaurant script, with slightly different events.

Script theory is attractive because it separates a fairly well defined, managable part of cognition. Scripts are intuitively plausible and well supported by experimental evidence (Bower et al., 1979). They also provide a computational theory, similar in spirit to the frame theory of representation in artificial intelligence (Minsky, 1981). The basic idea behind both is schematic knowledge: people organize knowledge about familiar objects, events and situations in terms of prototypes, or schemata. A schema contains representation for common knowledge, shared by all instances, and a number of slots which take different values for different instances. For example, a schema for a room

---

*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). Special thanks go to Michael Dyer for valuable comments on an earlier draft of this paper.

†To appear in *Connection Science* 2(1&2):83–101, 1990.

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

Table 1: **Representation of a script-based story as a causal chain and role bindings.**

contains the common structure (walls, floor, ceiling, door), and slots for size, shape, furniture, etc. A schema for a restaurant visit contains the common events (entering, seating, ordering, eating, etc.) and slots for the restaurant, food, amount of the tip, etc. (table 1).

Knowledge of scripts is important in natural language understanding because it allows for efficient communication. In narrative texts, many details are left out, because they can easily be filled in by the reader using scriptal knowledge. For example, in reading

`John went to fancy-restaurant. John asked the waiter for lobster. John left a big tip.`

the reader understands the last sentence by assuming the usual intermediate events in the restaurant script: the waiter brought John the lobster, John ate the lobster, the lobster tasted good, John paid the waiter, and John left a big tip (because the food was good). More complex texts contain references to several scripts, and understanding requires recognizing and applying several scripts at the same time. Scripts are used to connect the pieces of the text together, i.e. they are tools in building a complete representation of the story.

In symbolic artificial intelligence a script is represented as a causal chain of events with a number of open roles (table 1) (Schank and Abelson, 1977; Cullingford, 1978; DeJong, 1979; Dyer et al., 1987). Applying this knowledge to a story requires identifying the relevant script and filling in its roles with the actual constituents of the story. Once the script has been recognized and the roles have been instantiated, the sentences are matched against the events in the script. Events which were not mentioned in the story but are part of the causal chain can be inferred, as well as certain fillers of roles which were not specified in the story. For example, in the above story it is plausible to assume that John ate the lobster and the lobster tasted good.

The symbolic programs demonstrate the power of script theory in processing texts about everyday events. However, they do not address the question of where the scripts come from, i.e. how the knowledge of scripts can be learned from the experience. In symbolic systems, the script representations, their instantiation rules and inferences are hand-crafted with particular examples in mind.

A number of connectionist models of script application have also been proposed. In these models, the recognition of the appropriate script, its instantiation and inferences automatically emerge from the input story representation in a distributed fashion. Connectionist mechanisms for processing causal sequences (Golden, 1986), connecting multiple scripts (Sharkey et al., 1986), role binding (Dolan and Dyer, 1987), and learning inferences from examples (Miikkulainen and Dyer, 1989b; Miikkulainen, 1990) have been presented. However, the script representations are still coded in beforehand and remain fixed in these models.

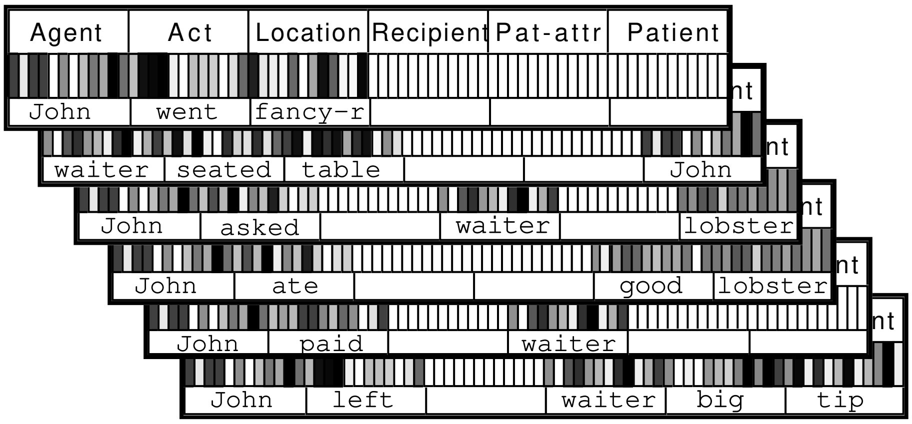

Figure 1: **Story representation.** A story is represented by a 360-dimensional vector of gray-scale values between 0 and 1. It consists of 6 sentence case-role representations, each $6 \times 10 = 60$ -dimensional. This particular story is an instance of fancy-restaurant track, with `PERSON = John, FANCY-FOOD = lobster`.

Scripts are regular event sequences, and it should be possible to learn these structures and their hierarchical relationships from the experience. This is how people develop new schemata. Scripts provide a means to organize experience.

The hierarchical feature map project has two main goals: (1) to show how the hierarchical taxonomy of scripts, tracks and role bindings can be extracted from examples of script-based stories, and (2) to show how the script taxonomy can serve as organization for the episodic memory. The stories in our system are examples of scripts, i.e. instantiations of the event sequence with specific role bindings. The taxonomy recognizes a story as an instance of a particular script, track and role binding and assigns a unique memory location for it. Incomplete story representations are automatically filled in as they are recognized.

2 Story examples

The example stories in our experiments were instances of the restaurant, shopping and travel scripts, each of which had three tracks, listed in the appendix. Two fillers (`John, Mary`) were used to fill the `PERSON` role in all tracks. In addition, the restaurant tracks had two fillers each for the `FOOD` roles, shopping tracks had two fillers each for the `ITEM` roles and travel tracks two fillers each for the `DESTINATION` roles. The actual input stories were generated by replacing the role names with different combinations of fillers. The script recognition system had to extract the hierarchical taxonomy from the repeated presentation of the resulting 36 story examples.

Each story in the training set was represented by the concatenation of the case-role representations of its sentences (figure 1). A case-role representation consists of assemblies of components, with distributed patterns in the assemblies indicating the fillers of the surface-semantic roles of the sentence (Hinton, 1981; Fillmore, 1968). The input for the feature map system thus consists of the *surface semantics of the sentences of the story.*

The case-role representations were formed by a separate front end, the backpropagation-based sentence parser network from the DISCERN system (Miikkulainen, 1990; Miikkulainen and Dyer, 1989b). This network reads an input sentence sequentially word by word and produces a case-role representation as its output. The network also develops distributed representations for the words while it is learning the case-role assignment task. The resulting representations reflect the regularities in the use of the words, and can be claimed to code the meanings of the words as well (Miikkulainen and Dyer, 1989a).

For each role, only a generic filler representation (e.g. `PERSON, FANCY-FOOD`) was developed by the sentence parser. Representations for the different filler instances (e.g. `John, Mary, lobster, steak`) were formed from the generic representation by replacing the first component with 0.0 for one and 1.0 for the other filler word. The purpose was to create two distinct instances with approximately the same surface semantics (Miikkulainen and Dyer, 1989a; Miikkulainen and Dyer, 1989b; Miikkulainen, 1990).

The word representation consisted of 10 units and the case-role representation of 6 assemblies (agent, act, location, recipient, patient-attribute and patient) (figure 1). Each story had 6 events, making the input vectors $6 \times 6 \times 10 = 360$ -dimensional. The input vector dimension is fixed because the number of input lines in the feature maps is fixed. This gives an upper bound to the number of events in the scripts, although shorter scripts are possible. The remainder of the vector is simply filled with zero.

The system was trained with complete script instantiations which included all events. A taxonomy of complete stories developed as a result. However, the recognition process is robust enough that only a few events need to be included to correctly classify a story. The representation for an incomplete story is formed by the front end, which places the events in their proper position in the input vector and fills the missing events with 0.5, the neutral value.

The script recognition system is not dependent on this particular type of story representation. The concatenation of event representations is perhaps the most straightforward approach. Another possibility would be to compose the vector from assemblies specifying the script name, track name, and the role bindings (Miikkulainen, 1990). It is essential, though, that the front end forms a stationary representation of the event sequence, which can then be input to the feature maps. Because the representation is not a sequence and is not explicitly causal, it can be said that the system recognizes the script as a single conceptual structure.

3 Self-organizing feature maps

The story representations are high-dimensional vectors with complex similarity relationships. With topological feature maps it is possible to lay out the representation space on a 2-D area so that the similarities between stories become visible.

A 2-D topological feature map (Kohonen, 1982b; Kohonen, 1984) implements a topology-preserving mapping from a high-dimensional input space onto a 2-D output space. The map consists of an array of processing units, each with N weight parameters (figure 2). The map takes an N-dimensional vector as its input, and produces a localized pattern of activity as its output. In other words, an input vector is mapped onto a location on the map.

Each processing unit receives the same input vector, and produces one output value. The response is proportional to the similarity of the input vector and the unit's weight vector. The unit with the largest output value constitutes the image of the input vector on the map. The weight vectors are ordered in such a way that the output activity smoothly decreases with the distance from the image unit, forming a localized response.

The ordering of the weight vectors retains the topology of the input space. This means roughly that nearby vectors in the input space are mapped onto nearby units in the map. This is a very useful property, since the complex similarity relationships of the high-dimensional input space become visible on the map (figure 3).

The organization of the map, i.e. the assignment of the weight vectors, is formed in an unsupervised learning process (Kohonen, 1982a; Kohonen, 1982c; Kohonen, 1984; Ritter and Schulten, 1988). Input items are randomly drawn from the input distribution and presented to the network one at a time (figure 2). The weight vector of the image unit and each unit in its neighborhood are

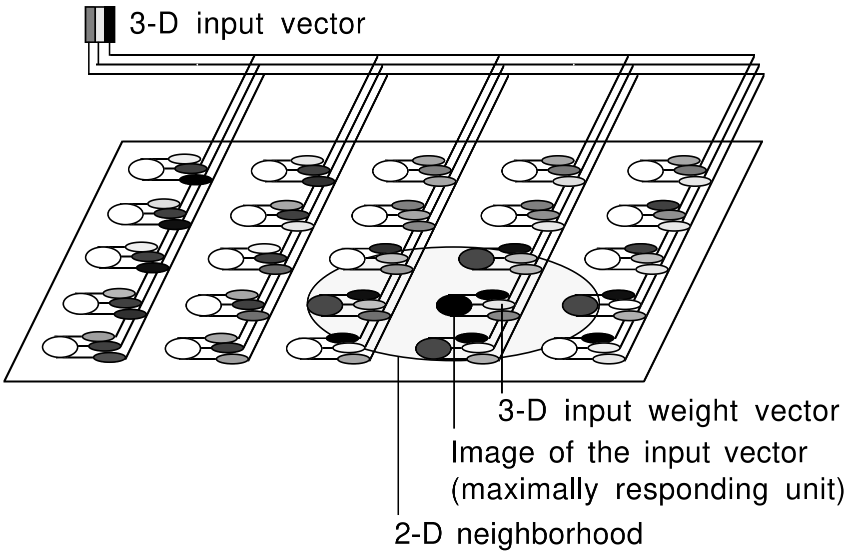

Figure 2: **A self-organizing feature map network.** A mapping is formed from a 3-dimensional input space onto a 2-dimensional network. The values of the input components, weights and the unit output are indicated by gray-scale coding.

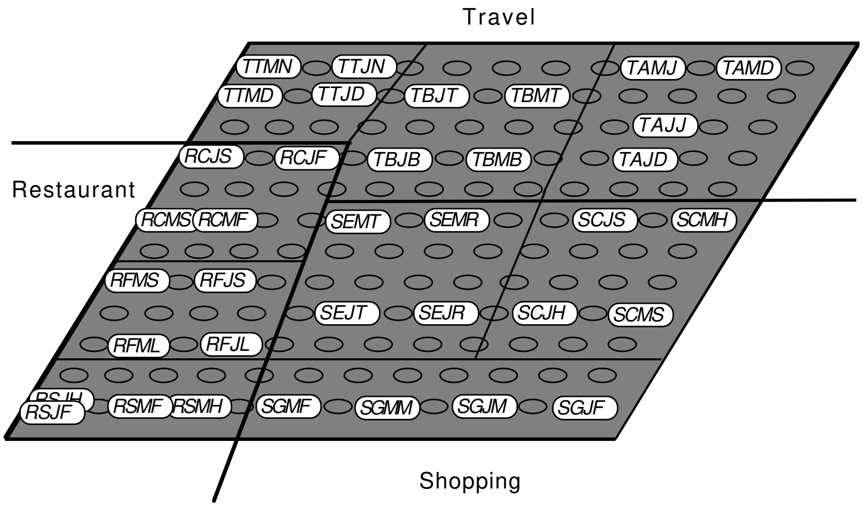

Figure 3: **Single-level mapping of script-based stories.** The first letter of a label indicates the script (`R=restaurant, T=travel, S=shopping`), the second letter stands for the track (`F=fancy-restaurant, C=coffee-shop-restaurant, S=fast-food-restaurant` etc.), third letter for the main actor (`J=John, M=Mary`), and the last for the two different bindings of a track-specific role (see description of the data in the appendix). Labels indicate the image of each story.

changed towards the input vector, so that these units will produce an even stronger response to the same input in the future. The parallelism of neighboring vectors is increased at each presentation, a process which results in a global order.

The process starts with very large neighborhoods, i.e. weight vectors are changed in large areas. This results in a gross ordering of the map. The size of the neighborhood decreases with time, allowing the map to make finer and finer distinctions between items.

There are several alternatives for implementing the similarity metric, neighborhood selection, and weight change. A biologically plausible process would be based on scalar products of the weight and input vectors, lateral inhibition and redistribution of synaptic resources (Kohonen, 1982c; Miikkulainen, 1987). These mechanisms can be abstracted and replaced with computationally more efficient ones without obscuring the process itself. The similarity can be measured by Euclidian distance, the neighborhood consists of the circular area around the maximally responding unit, and the weight changes are proportional to the Euclidian difference. More specifically, the weight components of unit (i, j) in the map are changed according to

$$\Delta\mu_{ij,k} = \begin{cases} \alpha(t)[x_k - \mu_{ij,k}] & \text{if } (i,j)\epsilon N_c(t) \\ 0.0 & \text{otherwise} \end{cases} \tag{1}$$

where $\mu_{ij,k}$ is the k:th component of the unit's weight vector, x is the input vector, $N_c(t)$ is the neighborhood around the maximally responding unit (shrinking with time), and $\alpha(t)$ as the gain of the adaptation.

In the self-organizing process, the distribution of the weight vectors becomes an approximation of the input vector distribution (Kohonen, 1982a; Kohonen, 1984). This means that the most frequent areas of the input space are represented to greater detail, i.e. more units are allocated to represent these inputs. The two dimensions of the map do not necessarily stand for any recognizable features of the input space. The dimensions develop automatically to facilitate the best discrimination between the input items.

Feature maps have several potentially useful properties for script recognition.

1. The classification performed by a feature map is based on a large number of parameters (the weight components), making it very robust. This suggests that incomplete or somewhat unusual event sequences can be correctly recognized.

2. The map is continuous, and can represent items between established categories. In other words, scripts can have soft boundaries.

3. The differences of the most frequent input items are magnified in the mapping, i.e. the variations of the most common event sequences are more finely discriminated.

4. The self-organizing process requires no supervision and makes no assumptions of the content of the input stories. The properties of the stories which best distinguish between scripts and their variations are determined automatically, and may be very different for different kinds of scripts.

On the other hand, scripts are hierarchical representations. Each script class consists of a number of tracks, and each track can be instantiated with different role bindings. Hierarchical feature maps can be used to make the hierarchical taxonomy explicit. The hierarchical architecture also cuts down the number of redundant system parameters (input weights) and concentrates the resources on the most salient aspects of the input data. The computationally intensive training is speeded up by effectively dividing the task into hierarchical subgoals.

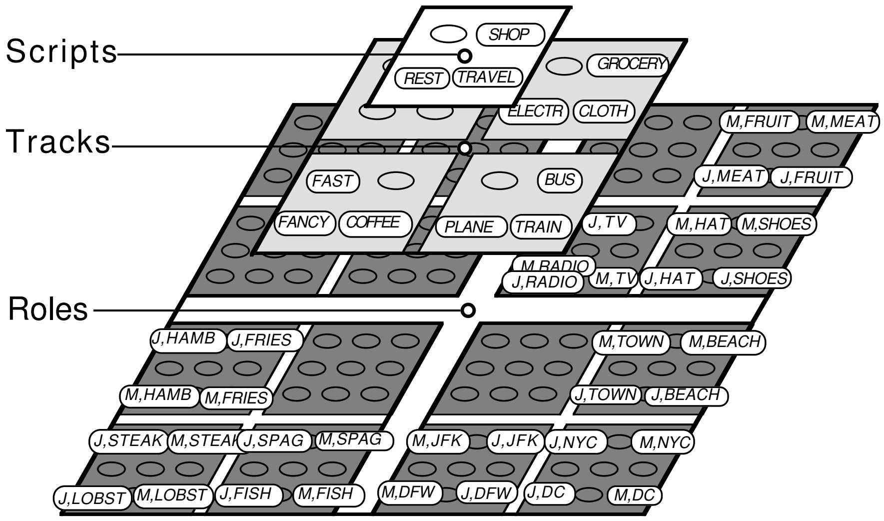

Figure 4: **Hierarchical feature mapping of script-based stories.** The highest level is a single 2×2 map, where the scripts are mapped onto different corners. The middle level consists of four 2×2 maps, which display the tracks of each script. The sixteen 3×3 maps at the lowest level map out the space of the possible role bindings for each track. The organization was formed in 40 training epochs.

4 Hierarchical feature maps

4.1 System architecture

The self-organizing process produces a representation of the input space where the input data is spread out spatially on e.g. a 2-dimensional sheet. The map represents most directly input spaces which are continuous and unstructured. If the data is hierarchical, the map reflects this through topological order. For instance, the map of taxonomical data essentially displays the minimal spanning tree of the data items, curved to fill in the whole area of the map (Kohonen, 1982b). Or, if we form a mapping of script-based stories, different variations of the scripts are mapped near each other (figure 3). Knowing what the hierarchical taxonomy of the data is, it is easy to see that the spatial layout of the map reflects the taxonomy. However, it is hard to *extract* the taxonomy from the map alone.

With hierarchical feature maps the taxonomy can be made explicit (figure 4). The hierarchy is a pyramid organized according to scripts, tracks and role-bindings. The highest level is a single feature map, which lays out the different script classes. Beneath each unit of this map there is another feature map, which lays out the tracks within the script. At the bottom level, the different role bindings within each track are separated. The map hierarchy receives a story representation as its input and classifies it as an instance of a particular script, track and role binding, indicated by the maximally responding units at each level of the hierarchy.

The highest-level map is laid out first in an ordinary self-organizing process. A small map size forces the process to make only a gross, high-level classification. A lower map in the structure receives as its input only those input items which belong to the category represented by its parent unit. In other words, the unit which "wins" the input item passes this item down to its submap. The lower map forms a subcategorization of these input items, mapping the differences within the script or the track. Each map displays topological order. This is most evident in the bottom level maps, where the `PERSON` role is differentiated along one axis, while the other axis separates the bindings of the other open role of the track. *These dimensions were discovered by the mapping itself, and they are different for different scripts.*

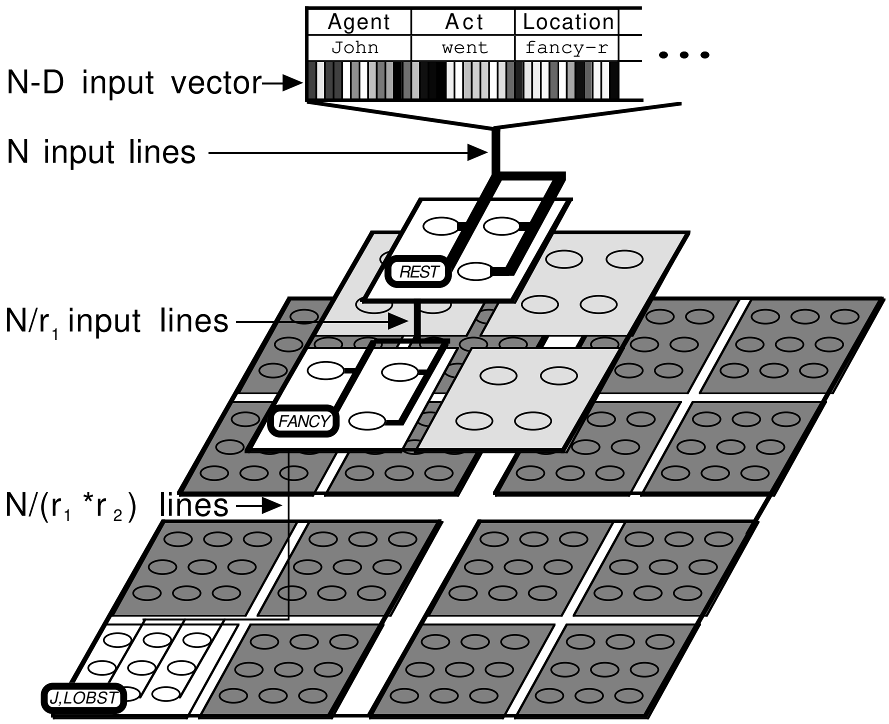

Figure 5: **Hierarchical feature map architecture.** Each unit independently determines the input lines with the most variance among its input vectors, and passes a compressed vector down to its submap for a more detailed mapping.

Input representations in a cognitive system often have some discrete structure, such as e.g. role specific assemblies (Hinton, 1981; McClelland and Kawamoto, 1986; Miikkulainen and Dyer, 1989a). If the data is hierarchical, the items belonging to the same category have a number of componets in common. These components can be dropped from the input to the next-level map (figure 5). The higher-level map acts as a filter, (1) choosing the relevant items for each submap and (2) compressing the representation of these items to the most relevant components before passing them on for a more detailed mapping.

Each unit has to determine independently which of its input lines are the most significant in distinguishing between items of the category. It has to find the *lines with the most variation* between the items it wins. These lines are different for different units. For example, a unit which stands for the fancy-restaurant track of the restaurant script wins only items which have `fancy-restaurant` as the location (figure 6). The values on the input lines which code the location do not change from one input to another (in our examples). On the other hand, the customer and food may be different in different stories, and lines representing these roles have a lot of variation. These lines alone are sufficient to determine the role bindings of the track, and they are therefore passed on to the lowest-level map.

John's visit to Leone's (figure 5). The top-level map receives the complete representation vector and maps it onto the unit labeled `REST`. This unit compresses the vector by removing the components whose values are the same in all restaurant stories (figure 6). The representation now consists of information which best distinguishes between the different restaurant stories. The `REST`-unit passes the compressed representation down to its submap, which classifies it as an instance of the fancy-restaurant track. Again, the `FANCY`-unit removes the components common to all fancy-restaurant stories, and passes the highly compressed vector to its submap. The representation is now limited to information about the role bindings, and it is mapped onto the unit representing

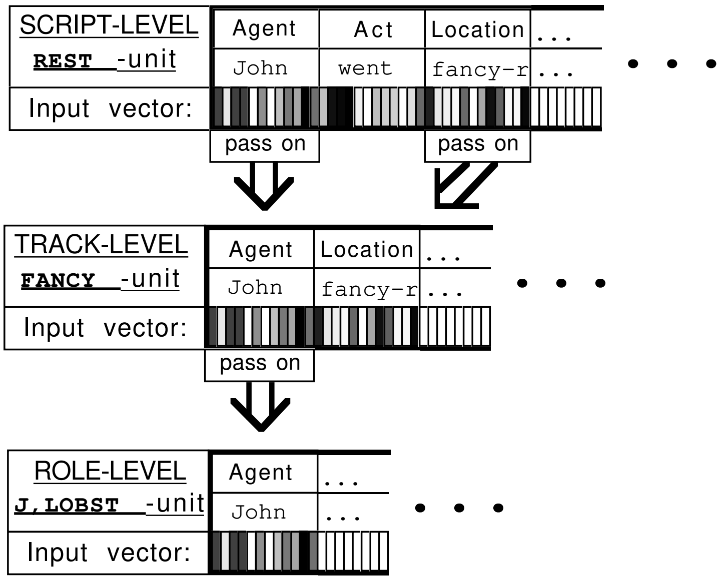

Figure 6: **Compression of the input lines.** The first few components of the story representation are shown. All stories mapped on the `REST`-unit share the act `went`, and these components can be dropped from further mapping. All inputs mapped on the `FANCY`-unit have `fancy-restaurant` as the location, and these components can be dropped. The input to the role level consists of only the filler of the customer -role.

customer=`John`, food=`lobster`.

The system can correctly classify a story even if a number of events are missing (filled with 0.5 in the representation). This is a consequence of the general redundancy of the system: the incomplete input is still closest to the correct script, and is classified as such.

4.2 Self-organization

In self-organizing a hierarchical feature map system, it is most efficient to start the process at the highest level, and include a lower level only when the higher level has become ordered. This way the set of inputs to the lower level stays constant, and no work is wasted on inputs which will eventually be classified into another category. 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. A story representation is first classified as an instance of a script, then as a track within the script etc. Within the script (or track), *only a subset of the input representation needs to be examined to refine the classification further.* The relative differences between the items are greater, making the self-organization easier.

In a single-level map all units need to receive the complete representations of all input items (figure 2). Modifying the weights is costly, because the initial neighborhood must cover almost the entire map (Miikkulainen, 1987). In a hierarchical map system, the maps at lower levels receive only small subsets of the original input lines, and the neighborhoods are always small because the maps are small at all levels (figure 5). This reduces the training time considerably.

Each unit determines its own compressed set of input lines during the last epoch before the lower level is included. In our implementation, the lines of each unit were simply ordered with respect to the variance over the inputs won during that epoch, and a fixed fraction r_i was chosen at each map level i (figure 5). This means that usually only a few lines of the representation for e.g. customer were passed on to the next level (the ones with the highest variance), not the whole representation. Instead of a fixed fraction, a variance threshold could also be used, in which case the extent of the compression would adapt to the input data.

Of the 360 input lines to the first level, each script unit independently determined the 54 (15%) lines with the highest variance, and passed them on to the second level. This compressed vector consisted mostly of the differences in the order of events in the track, and of the most unique events. Of these 54 lines, each second-level unit passed on 10 (20%) to the lowest level. These lines consisted mainly of the first components of the filler words (either 0.0 or 1.0), which best differentiate between fillers.

The outcome of the self-organizing process is somewhat sensitive to the system parameters. To obtain the three-level classification into scripts, tracks and role bindings the maps must have approximately the right size and each input story must be approximately as frequent in the input data. If, for instance, the highest-level map is too large, or one of the script classes is far more frequent than the others, the different tracks may get separated already at the highest-level map. Even if the parameter settings are not ideal, the script recognition works the same. *The system uses whatever architecture is given to establish the best classification of the input data.* The configuration parameters may need to be experimentally adjusted to achieve the desired semantics for the levels of the hierarchy, but the function of the recognition system is fairly robust.

5 Reproducing stories

It is possible to reproduce the story from its representation in the feature maps. In the self-organizing process the input weight vectors become approximations of the input vectors (Kohonen, 1982a; Kohonen, 1984). The weight vector at each unit represents an average of all stories mapped on that unit. The weights on constant input lines accurately represent the constant components, while the weights on the input lines with high variance represent averages. The high-variance lines were passed down in the hierarchy for a more accurate mapping, and the different values for these components are accurately represented at the lower levels of the hierarchy. A complete and accurate story representation can be retrieved by concatenating the accurate weights, i.e. weights on the compressed input lines, of the image units at all levels (figure 7).

This technique can be used to produce a full paraphrase of an incomplete story, i.e. *to fill in the missing events.* The story is first classified as an instance of one of the stories known to the system. As discussed above, the system can usually determine the correct script, track and role binding even if number of events are missing. The weights for this classification are then read out, forming the full paraphrase. For example, the story

`John went to fancy-restaurant. John asked the waiter for lobster. John left a big tip.`

is classified as an instance of the restaurant script, fancy-restaurant track, with `PERSON` = `John`, `FANCY-FOOD` = `lobster`. Concatenating the weights on the compressed input lines gives the full instantiation of the script:

`John went to a fancy-restaurant. Waiter seated John at a table. John asked the waiter for lobster. John ate the good lobster. John paid the waiter. John left the waiter a large tip.`

Once the story has been recognized, the bindings are consistent and plausible in the weight representation. In other words, the recognition system implements plausible script-based inferencing (Dyer, in press).

The weight vector representations obtained from the different levels of the hierarchy represent *different levels of abstraction.* A top-level unit stands for the skeleton of the script where the different tracks and role bindings have been averaged out. When the compressed top-level weights are combined with the track-level weights, the track becomes established but the role bindings

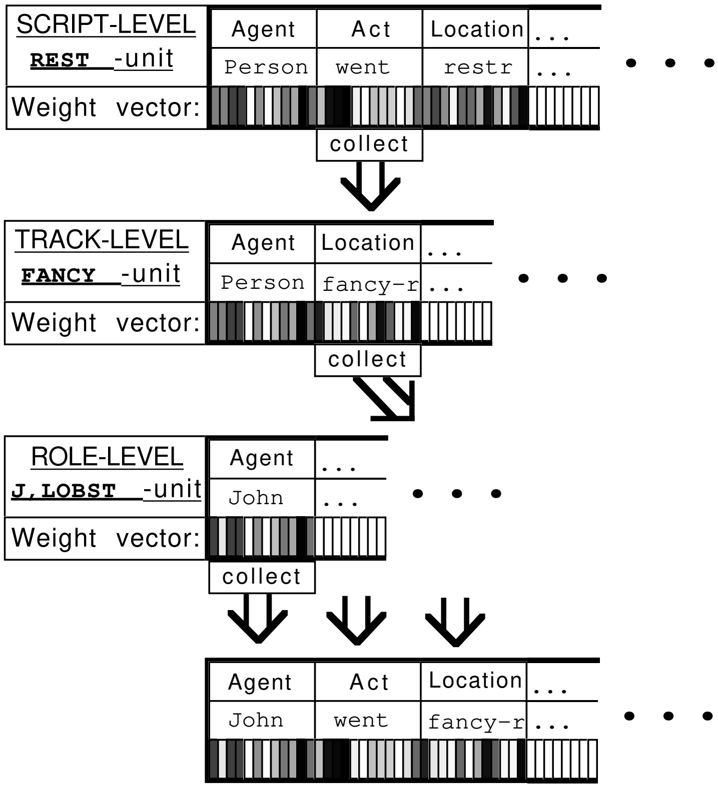

Figure 7: **Reproducing stories.** The complete story representation is obtained by concatenating the weights on the compressed input lines at each level.

are still unspecified. For example, in the place of the agent name (`John` or `Mary`) there is the representation of the general agent (`PERSON`), which is an average of the two (figure 7). The bindings are finally established when the lowest-level weights are combined with the representation.

The maps contain several units which are not images of any particular input item. These extra units are needed in the self-organizing process to develop a meaningful topological organization of the map. They constitute extra space that makes the mapping continuous. The weights on these units represent stories which are *combinations of the actual inputs* that the system has seen. In some cases they may be meaningful abstractions or generalizations, extracted from the similarities of the items around them in the map. Such a unit between two script units, for example, can represent uncertainty about the script class the incomplete input story belongs to. Some events can be safely filled in, and they are accurately represented in the weights of this unit. Others are uncertain, and the corresponding weights represent averages of the different possibilities. In many cases though, the unlabeled units do not represent anything useful. A combination of e.g. the airplane-travel script and the cake-baking script is not anything that occurs in real life. However, in a system consisting of a large number of scripts, these scripts would probably be mapped on different areas in the map. The map tends to maximize continuity, and most unlabeled units lie between scripts that are similar, and whose combinations are possible.

6 Episodic memory organization

The hierarchical feature maps form a recognition taxonomy for script-based stories. The taxonomy can also be used to organize an episodic memory of these stories (Miikkulainen, 1990). Stories are read in and memorized one at a time and can be retrieved with a partial cue.

An ordinary feature map is a classifier, mapping an input vector to a location on the map. A trace feature map (Miikkulainen, 1990), in addition, creates a memory trace on that location. The map remembers that at some point it received an input item that was classified at that location. Trace feature map mechanism is used at the bottom level of the map hierarchy to create a trace of the story.

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 creates a memory trace at that location using lateral connections between units.

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 reasonable guess about the role binding. The trace map mechanism then determines whether an appropriate trace exists and settles into the accurate role binding if one is found. The complete story representation is retrieved from the weight vectors of the maximally responding units at the script, track, and role binding levels.

The episodic memory for the script-based stories has two requirements which are not met by most neural network models of associative memory (Kohonen, 1984; Hopfield, 1982):

1. A few representation components, i.e. the ones discriminating between the different instances, need to be stored and retrieved with extreme accuracy.
2. The representations need to be stored "one-shot", with only a single presentation, without knowledge about additional traces to be stored.

The hierarchical feature maps employ abstractions to separate crucial information (instances, mapped at the bottom level) from information which is more widely distributed and where accuracy is not as critical (components common to the script and track, at higher levels). The maps assign different areas of the memory to different items, making it possible to store them one-shot.

As was pointed out in (Schank and Abelson, 1977), people seem to make scripts out of all regularities in everyday experiences, and in several levels of complexity, abstraction and extent. The hierarchical feature map system can be seen as a general model of the episodic memory, similar in spirit to the symbolic model of (Kolodner, 1983a; Kolodner, 1983b). Incoming experiences are classified in terms of their similarities to previous experiences (higher-level maps), and stored in the memory in terms of their differences (bottom level maps). In doing so, the structure of the memory also changes, and affects future classification. The structure is extracted autonomically from examples, and reflects the moving statistical regularities in the data.

7 Discussion

With hierarchical feature maps, it is possible to recognize and represent a large number of different scripts, and the system determines automatically what is relevant for the taxonomy. The stories are classified into script classes, tracks, and different role bindings based on the statistical similarities in the input representations. The breakdown of each script is independent of the others. Tracks and roles are developed which facilitate the best discrimination between the instantiations of the script, and these are usually different for different scripts.

On the other hand, the classification is purely statistical. There is no way to include semantic information to guide the process, as in the supervised learning approach. All relevant information must be included in the input vectors. Also, since the relevant roles are determined separately for each script, the common roles end up being represented multiple times. Each script in our data has a role for the main actor, but this role has to be mapped separately on every map at the lowest level. Half of each role map is dedicated to stories with `John` as the main actor, the other half with `Mary` (figure 4).

The self-organizing process resembles a human learner in building the script taxonomy. The process begins by establishing a gross ordering of the input data, dividing the input into a few large categories (Miikkulainen, 1987). The most prominent and regular event sequences are recognized as the first rudimentary scripts. For example, the restaurant and shopping stories are first grouped together, separate from the travel stories. These categories become gradually more refined, as more attention is paid on the details. *The more frequently a certain kind of input occurs in the input data, the more of its details become significant.* The restaurant and shopping stories together are twice as numerous as the travel stories, and what used to be different variations of the shopping-restaurant script eventually become scripts on their own right.

The number of units e.g. in the person-dimension at the bottom level maps poses an upper limit on the number of different person names the system can tell apart. However, the system need not see all these names in the training data. If continuous ID values are used for the different person representations (in the above data, ID was the first unit in the representation, e.g. 0.0 for `John` and 1.0 for `Mary`), a statistical sample is sufficient to form a continuous mapping of the possible ID values. In other words, the whole space of possible fillers can be covered with limited training data, but the resolution is limited by the number of units in the map.

The hierarchical feature map technique actually gives us a continuum of classification systems. At one extreme there is the single-level feature map, which produces a topological layout of the input space. At the other extreme there is a hierarchy of two-unit maps, which forms a statistically balanced divisive clustering tree of the input items: the inputs are first divided into two equally large classes, each class is further divided into two parts etc.

The hierarchical feature mapping technique makes most sense when the input data is strongly hierarchical, and the architecture of the system matches this hierarchy. In the extreme case of a uniform distribution of independent continuous variables there is nothing to be gained by using hierarchies. If enough is known about the structure of the input space, *a hierarchical feature map architecture can be chosen, which self-organizes to directly represent the hierarchical semantics of the input space.*

If the input data has enough structure, the reduction of the required input connections and the speed-up of the self-organization can be quite dramatic. In the script recognition system, there were $4 \times 360 = 1,440$ connections to the highest level, $16 \times 54 = 864$ to the second, and $144 \times 10 = 1,440$ to the bottom level, a total of 3,744 input connections. A comparable single-level mapping (figure 3) with 144 units for the same data required 51,840 input connections. Self-organizing the single-level mapping took about 9.5 hours on an HP 9000/350 workstation, while the hierarchical mapping was complete in three minutes, a difference of two orders of magnitude. In this serial simulation, one order of magnitude speed-up directly results from the one order of magnitude reduction in the input connections. Another order of magnitude speed-up comes from the hierarchical subgoaling approach, and applies to fully parallel implementations as well.

8 Future work

It might be possible to develop a mechanism for automatically adjusting the sizes of the maps or the depth of the hierarchy according to the input data. Two different inputs should always be mapped onto different units at the lowest level. This constraint could be used to self-organize the system architecture, in addition to the current self-organization of the individual feature maps.

It would be desirable to develop a more general representation for the input stories. A large number of units in the current representation are always blank, and the representation is extremely sensitive to the order and number of events. Instead of simple concatenations of a fixed number of event representations, the system should be able to read sequences of event representations of undetermined length. One possible approach is to use a feature map of events as a front end. The events would be presented to this map sequentially, and would form a trajectory on the map. This trajectory (i.e. the composite response pattern on the front end map) would form the input to the script recognition system.

The primary function of the system is to build a taxonomy which allows recognizing a story as an instance of a particular script, track and role binding. Higher-level systems such as a paraphraser or a question answerer could be built based on this representation. It might be possible to extract the information the self-organized system has found about the relevant tracks and roles, and use this to automatically generate training data for supervised-learning based higher-level systems, thus combining the advantages of both approaches.

9 Conclusion

The hierarchical feature map system shows how the hierarchical taxonomy of scripts, tracks and role bindings can be extracted from experience, and how the taxonomy can be used to recognize and fill in incomplete instantiations of the script. Hierarchical feature maps also have a number of properties which make them useful for memory organization: (1) the maps visualize the data very nicely, with the hierarchy displaying the script taxonomy and the maps laying out the topology at each level, (2) the maps are continuous and can represent items between established categories, (3) variations of the most common inputs are more finely discriminated and (4) the most salient aspects of the input data are separated and most resources are concentrated on them.

References

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

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

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

Dolan, C. P. and Dyer, M. G. (1987). Symbolic schemata, role binding, and the evolution of structure in connectionist memories. In *Proceedings of the IEEE First International Conference on Neural Networks.* Piscataway, NJ: IEEE.

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

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

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

Golden, R. M. (1986). Representing causal schemata in connectionist systems. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum.

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

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences, USA*, pages 2554–2558.

Kohonen, T. (1982a). Analysis of a simple self-organizing process. *Biological Cybernetics*, 44:135–140.

Kohonen, T. (1982b). Clustering, taxonomy, and topological maps of patterns. In *Proceedings of the Sixth International Conference on Pattern Recognition*. IEEE Computer Society Press.

Kohonen, T. (1982c). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69.

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

Kolodner, J. L. (1983a). Maintaining organization in a dynamic long-term memory. *Cognitive Science*, 7:243–280.

Kolodner, J. L. (1983b). Reconstructive memory: A computer model. *Cognitive Science*, 7:281–328.

McClelland, J. L. and Kawamoto, A. H. (1986). Mechanisms of sentence processing: Assigning roles to constituents. In McClelland, J. L. and Rumelhart, D. E., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 2: Psychological and Biological Models*. Cambridge, MA: MIT Press.

Miikkulainen, R. (1987). Self-organizing process based on lateral inhibition and weight redistribution. Technical Report UCLA-AI-87-16, Los Angeles: Computer Science Department, University of California, Los Angeles.

Miikkulainen, R. (1990). A neural network model of script processing and memory. Technical Report UCLA-AI-90-03, Los Angeles: Computer Science Department, University of California, Los Angeles.

Miikkulainen, R. and Dyer, M. G. (1989a). Encoding input/output representations in connectionist cognitive systems. In Touretzky, D. S., Hinton, G. E., and Sejnowski, T. J., editors, *Proceedings of the 1988 Connectionist Models Summer School*. San Mateo, CA: Morgan Kaufmann.

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

Minsky, M. (1981). A framework for representing knowledge. In Haugeland, J., editor, *Mind Design: Philosophy, Psychology, Artificial Intelligence*. Cambridge, MA: MIT Press.

Ritter, H. J. and Schulten, K. J. (1988). Convergency properties of Kohonen's topology conserving maps: Fluctuations, stability and dimension selection. *Biological Cybernetics*, 60:59–71.

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

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

Appendix: Input stories

The data consisted of restaurant, shopping and travel scripts, with three tracks each. Each track had two open roles. Two fillers (`John, Mary`) were used to fill the `PERSON` role in all tracks. The fillers for the other open role (`FOOD, ITEM` or `DESTINATION`) are listed next to the track name below.

RESTAURANT SCRIPT

Fancy-restaurant track
FANCY-FOOD = lobster, steak

```
PERSON went to a fancy-restaurant.
Waiter seated PERSON at a table.
PERSON asked the waiter for FANCY-FOOD.
PERSON ate the good FANCY-FOOD.
PERSON paid the waiter.
PERSON left the waiter a large tip.
```

Coffee-shop-restaurant track
COFFEE-FOOD = spaghetti, fish

```
PERSON went to a coffee-shop-restaurant.
PERSON sat down at a table.
PERSON asked the waiter for COFFEE-FOOD.
PERSON ate the COFFEE-FOOD.
PERSON paid at the cashier.
PERSON left the coffee-shop-restaurant.
```

Fast-food-restaurant track
FAST-FOOD = hamburger, fries

```
PERSON went to a fast-food-restaurant.
PERSON waited in line for the cashier.
PERSON asked the cashier for FAST-FOOD.
PERSON paid the cashier.
PERSON ate the FAST-FOOD.
PERSON left the fast-food-restaurant.
```

SHOPPING SCRIPT

Clothing-shopping track
CLOTHING-ITEM = shoes, hat

```
PERSON went to a clothing-store.
PERSON looked for a CLOTHING-ITEM.
PERSON compared CLOTHING-ITEM prices.
PERSON tried a number of CLOTHING-ITEMs.
PERSON chose the best CLOTHING-ITEM.
PERSON paid at the cashier.
```

Electrical-shopping track
ELECTRICAL-ITEM = TV, radio

```
PERSON went to an electrical-store.
PERSON asked the cashier for an ELECTRICAL-ITEM.
PERSON asked questions about the ELECTRICAL-ITEM.
PERSON compared ELECTRICAL-ITEM prices.
PERSON chose the best ELECTRICAL-ITEM.
PERSON paid the cashier.
```

<u>Grocery-shopping track</u>
GROCERY-ITEM = fruit, meat

```
PERSON went to a grocery-store.
PERSON chose a shopping-cart.
PERSON chose a number of GROCERY-ITEMs.
PERSON compared GROCERY-ITEM prices.
PERSON waited in line for the cashier.
PERSON paid the cashier.
```

<u>TRAVEL SCRIPT</u>

<u>Airplane-travel track</u>
PLANE-DESTINATION = JFK, DFW

```
PERSON checked-in at the airport.
PERSON waited for the boarding.
PERSON got-on the plane.
The plane took-off from the airport.
The plane arrived at the PLANE-DESTINATION.
PERSON got-off the plane.
```

<u>Train-travel track</u>
TRAIN-DESTINATION = NYC, DC

```
PERSON bought a ticket at the railway-station.
PERSON waited for the train.
PERSON got-on the train.
The conductor punched the ticket.
The train arrived at the TRAIN-DESTINATION.
PERSON got-off the train.
```

<u>Bus-travel track</u>
BUS-DESTINATION = town, beach

```
PERSON went to the bus-stop.
PERSON waited for the bus.
PERSON got-on the bus.
PERSON paid the driver.
The bus arrived at the BUS-DESTINATION.
PERSON got-off the bus.
```