SICONLP90

Distributed Connectionist Knowledge Representations in Script/Goal-Based Story Understanding

Geunbae Lee, Risto Miikkulainen*
Artificial Intelligence Laboratory
Computer Science Department
University of California, Los Angeles, CA 90024, U.S.A.
gblee@cs.ucla.edu, risto@cs.ucla.edu; (213)-825-5199

Abstract

This paper describes construction of *distributed semantic representations* (DSRs) which have many symbol-like properties while retaining the nice features of PDP (Parallel Distributed Processing) representations. DSRs are formed by encoding structured objects as triples in auto-associative recurrent PDP networks. DYNASTY is a distributed connectionist model of story paraphraser, which reads script/goal-based stories and generates full inference chains describing the stories.

1 Background and Issues

Issues in distributed/holographic and localist/punctate representation of high-level knowledge have received a lot of attention recently. While [Feldman, 1986] has given arguments against both extreme punctate and extreme holographic representations, PDP researchers, such as [Rumelhart et al., 1986b], have listed numerous advantages that distributed representations have over localist representations. At the same time, a number of techniques have been developed for forming distributed representations, including back propagation [Rumelhart et al., 1986a] and extended back-propagation [Miikkulainen and Dyer, 1988], conjunctive and coarse codings [Hinton et al., 1986], microfeature-based representations [Waltz and Pollack, 1985; McClelland and Kawamoto, 1986], and tensor product representations [Dolan and Smolensky, 1989; Smolensky, 1987].

Forming distributed representations which are able to support higher-level reasoning and represent conceptual knowledge is not an easy task. Whereas the "von Neumann symbol" -approach starts with random bit strings (like ASCII code) and builds structural relationships between symbols to represent conceptual knowledge, the distributed (or so-called "subsymbolic" [Smolensky, 1988]) representation ought to implement both structure and semantics below the symbolic level as a pattern in an ensemble of neuron-like elements.

2 Criteria for a Distributed Semantic Representation

A distributed representation that is able to represent conceptual knowledge must have four properties:

(1) *Automaticity* – The representation must be acquired through an automatic learning procedure, rather than be set by hand. For instance, the hand-coded microfeature-based representation used in [McClelland and Kawamoto, 1986] does not meet this criterion.

(2) *Portability* – The representation should be global rather than locally confined to its training environment. That is, the representation learned in one training environment must be portable to another task environment. For example, the representation in Hinton's family tree example [Hinton, 1986] can be said to meet the automaticity criterion, but not the portability criterion, since it cannot be used in any other task.

*Current address: Department of Computer Sciences, The University of Texas at Austin, risto@cs.utexas.edu

(3) *Structure encoding* – [Feldman, 1986] has argued that any conceptual representation must support answering questions about the structural aspects of the concept. For example, part of the meaning of "irresponsible" is that there was an obligation established to perform an action and the obligation was violated. To answer a question about the meaning of "irresponsible" requires accessing these constituent structures. Any conceptual representation must have structural information in the representation itself about the constituents of the concept. Purely holographic representations alone do not meet this criterion.

(4) *Similarity-based representations* – Distributed representations gain much of their power by encoding statistical correlations in the training data, and thus forming a characterization of the environment. The statistical correlations give connectionist models their ability to generalize. To support generalization, distributed representations should exhibit semantic content at the micro level, i.e. similar concepts should have similar distributed representations. This criterion provided the original inspiration for microfeature-based encoding. Representations for similar concepts are similar because they share similar microfeatures.

3 Forming Distributed Semantic Representations (DSRs) of Words

In this section we show how DSRs are formed and demonstrate their validity for the task of encoding word meanings.

There are two alternative views on the semantic content of words: (1) The structural view defines a word meaning in terms of its relationships to other meanings. (2) The componential view defines meaning as a vector of properties (e.g. microfeatures). We take an interim view – that meaning can be defined in terms of a distributed representation of structural relationships, where each relationship is encoded as a proposition. Examples of propositions are verbal descriptions of action-oriented events in everyday experiences.

3.1 Representing DSRs

The intuition behind DSRs is based on our observation that people sometimes learn word meanings through examples of their relationships to other words. For example, after reading the 4 propositions below, the reader easily forms an idea of the meaning of "foo".

- proposition1: The man drinks foo with a straw.
- proposition2: The company delivers foo in a carton.
- proposition3: Humans get foo from cows.
- proposition4: The man eats bread with foo.

Apparently, "foo" means something similar to "milk". An interesting fact is that the meaning of "foo" is not fixed, rather it is gradually refined as the reader experiences more propositions in varying environments. To develop DSRs based on propositions, we have to define the structural relationships between concepts with respect to those propositions. For action-oriented events describing propositions, we use thematic case-role representation. The case-role theory was originally developed by [Fillmore, 1968], and it has been extended in natural language processing systems [Schank and Riesbeck, 1981]. For example, the DSR of "milk" can be defined as the composition of case-roles using these 4 propositions:

milk = F_i (G_c (object, *proposition1*), G_c (object, *proposition2*), G_c (object, *proposition3*), G_c (co-object, *proposition4*),.....)

where *milk* is the representation of the meaning of "milk"; F_i is a function that integrates all the propositions involving the concept of milk, and G_c is a function combining the case-role relationships into the propositions. In the same way, each proposition is temporally defined as the composition of the constituent case-role components such as:

proposition1 = F_i (G_c (agent, *man*), G_c (verb, *drink*), G_c (object, *milk*), G_c (instrument, *straw*))

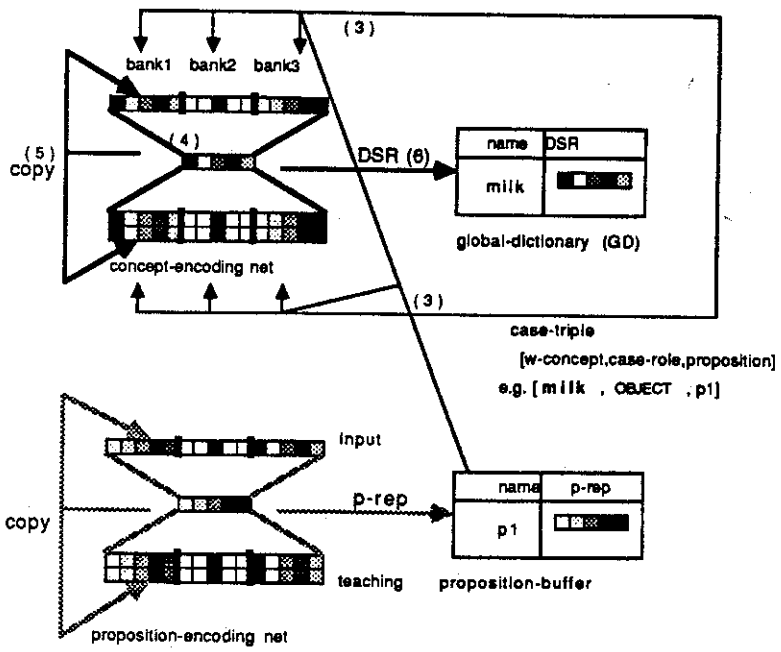

Figure 1: **Two XRAAM architectures for learning DSRs.** Each network has an input, hidden, output, and teaching layer. Once a concept (or proposition) in the triple form has been auto-associated, the pattern of activation in the hidden-layer units is stored in the corresponding global dictionary (or proposition buffer) as the representation of the concept (or proposition). The concept-encoding network forms a representation of a symbol by encoding all propositions involved in representing that symbol, while the proposition-encoding network forms a representation of a proposition by encoding all the symbols involved in representing that proposition. Thus, symbols in the global dictionary are fed to the input/output layers of each XRAAM. In this figure, the dark lines designate the data flow in the concept-encoding procedure.

3.2 Learning DSRs

We have developed XRAAMs for automatically learning DSRs. XRAAMs (extended recursive auto-associative memory) are based on Pollack's RAAMs [Pollack, 1988], but have external memory to store the developed internal representations. The basic idea is to "re-circulate" the internal representation (hidden layer of the network) back out to the environment (input and output layers of the network). This idea has been suggested in various forms, e.g. as FGREP [Miikkulainen and Dyer, 1988], Recursively Reduced Descriptions [Hinton, 1988], Recursive Auto-Associative Memories [Pollack, 1988], Sequential Connectionist Networks [Jordan, 1986; Elman, 1988], and it has been used in natural language question-answering [Allen, 1988], parsing [Hanson and Kegl, 1987] and sentence comprehension [St. John and McClelland, 1989]. Figure 1 shows two XRAAM modules in our story processing architecture.

The learning portion of the XRAAM architecture contains two symbolic memories (Global Dictionary and Proposition Buffers) and two 3-layer ARPDP (auto-associative recurrent PDP) networks. The input and output layers of each network have 3 banks of units (figure 1). After all 3 banks have been properly loaded, the DSR emerges in bank1 as a result of an auto-associative BP (Backward Propagation) process [Rumelhart et al., 1986a].

The DSR learning process consists of two alternating cycles: Concept Encoding and Proposition Encoding. Below we informally describe each cycle. In each cycle, all concept and proposition representations start with a "DON'T CARE" pattern, e.g. 0.5, when the activation value range of each unit in network is 0.0 to 1.0. The case-role representation is fixed using orthogonal bit patterns (for minimizing interference).

Concept Encoding Cycle:

1. Pick one concept to be represented, say CON1.
2. Select all relevant triples for CON1. In the *milk* example, they would be triples like (*milk* object proposition1), (*milk* object proposition2), (*milk* object proposition3), etc. For the first triple, load the initial representation for CON1 into bank1.

3. Load the case-role representation into bank2, and load its corresponding proposition into bank3. In the *milk* example, for the first triple, bank1, bank2, and bank3 are loaded with bit patterns for *milk*, object, proposition1 respectively.

4. Run the auto-associative BP algorithm, where the input and output layers have the same bit patterns.

5. Re-circulate the developed (hidden layer) representation into bank1 of both the input/output layers and perform step3 to step5 for another triple until all triples are consumed.

6. Store the developed DSR into the global dictionary and select another word concept to be represented.

Proposition Encoding Cycle: Basically this cycle undergoes the same steps as the Concept Encoding Cycle, except that now we load bank1, bank2, and bank3 with the proposition to be represented, the case-role, and its corresponding concept representation (DSR). The result of the encoding is stored into the proposition buffer which can be flushed and reused after we acquire all the necessary stable bit patterns for all concepts.

Now the overall DSR learning process will be:

1. Perform the entire concept encoding cycle.

2. Perform the entire proposition encoding cycle.

3. Repeat step1 and step2 until stable patterns are obtained for all concepts.

In this process,the integration function F_i is embedded into the dynamics of the Recursive Auto-Associative Stacking operation [Pollack, 1988] and the combination function G_c is a concatenation of two bit patterns.

3.3 Decoding DSRs into the constituents

The decoding process is the reverse of encoding: We load the concept representation in the hidden layer of the concept encoding network and perform value propagation until we get the desired case-role in bank2 and proposition in bank3 of the output layer. Next, we load the resulting proposition in the hidden layer of the proposition encoding network and get back the constituent case-roles and concept representations. Figure 2 shows the decoding architecture.

3.4 Experiment: Learning DSRs for Nouns and Verbs

We conducted a number of experiments to see how well XRAAM networks learn DSRs for nouns and verbs. Proposition generators similar to the ones used in [McClelland and Kawamoto, 1986] were used to generate 60 propositions. Each category in the generators was replaced by proper fillers for each proposition. We analyzed each proposition's case structure in order to load them into our network architecture. Table 1 shows proposition generators with their case structures and table 2 shows concept categories with their fillers.

In this simulation, both the concept encoding network and the proposition encoding network have a 30 unit input layer (each bank has 10 units), a 10 unit hidden layer, and a 30 unit output layer. The DSRs and proposition representations are 10 units. Figure 3 shows the DSRs developed for a number of nouns and verbs. These are snapshots of 120 epochs, where one epoch is 200 cycles of autoassociative BP for each concept and proposition. Notice that the learned representations show similarities according to the concept categories. Words in the same semantic category have similar representations because they behave similarly in the given propositions. Interestingly, words with multiple categories (e.g. dog) develop representations distinct from words with a single category (e.g. wolf). This is because words with multiple categories can be considered to have multiple "usages". For example, the word "dog" is used as both the AGENT and CO-OBJECT in the proposition generators.

In order to see the similarities more clearly, we have run the merge clustering algorithm [Hartigan, 1975] on the learned DSRs. Figure 4 shows the clustering analysis results. The DSRs belonging to the same categories are merged early in the process.

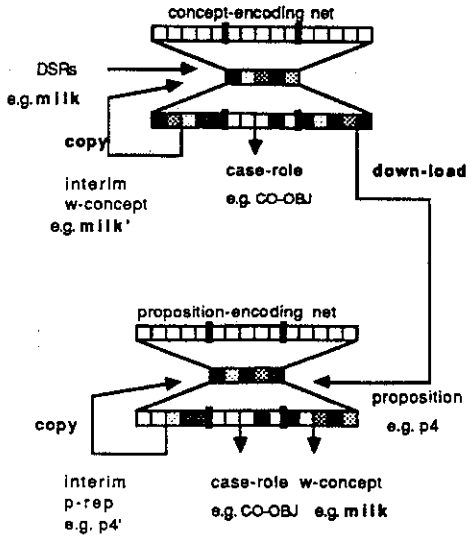

Figure 2: **Decoding Architecture for DSRs.** The examples illustrate the propositions in the text (see section 3.1). The decoding sequence proceeds from the root of the tree to the leaves (the opposite of encoding).

P numb.	P Gen.	Case Structures
p1	human ate	AGENT-VERB
p2	human ate food	AGENT-VERB-OBJECT
p3	human ate food with food	AGENT-VERB-OBJECT-COOBJ
p4	human ate food with utensil	AGENT-VERB-OBJECT-INST
p5	animal ate	AGENT-VERB
p6	human broke fragile-object	AGENT-VERB-OBJECT
p7	human broke fragile-object with breaker	AGENT-VERB-OBJECT-INST
p8	breaker broke fragile-object	INST-VERB-OBJECT
p9	animal broke fragile-object	AGENT-VERB-OBJECT
p10	fragile-object broke	OBJECT-VERB
p11	human hit thing	AGENT-VERB-OBJECT
p12	human hit human with possession	AGENT-VERB-OBJECT-COOBJ
p13	human hit thing with hitter	AGENT-VERB-OBJECT-INST
p14	hitter hit thing	INST-VERB-OBJECT
p15	human moved	AGENT-VERB
p16	human moved object	AGENT-VERB-OBJECT
p17	animal moved	AGENT-VERB
p18	object moved	OBJECT-VERB

Table 1: **Proposition generators.** The proposition generators are presented with their proposition numbers and case structures. Each category slot (e.g human) can be filled with any of the concepts in the table 2 (e.g. man). The OBJECT role in the case structures is different from the category name "object" in the proposition generators.

Categories	Concept Fillers
human	man, woman
animal	dog, wolf
object	ball, desk
thing	human, animal
food	cheese, spaghetti
utensil	fork, spoon
fragile-object	plate, window
hitter	ball, hammer
breaker	hammer, rock
possession	ball, dog

Table 2: **Categories and their filler concepts.**

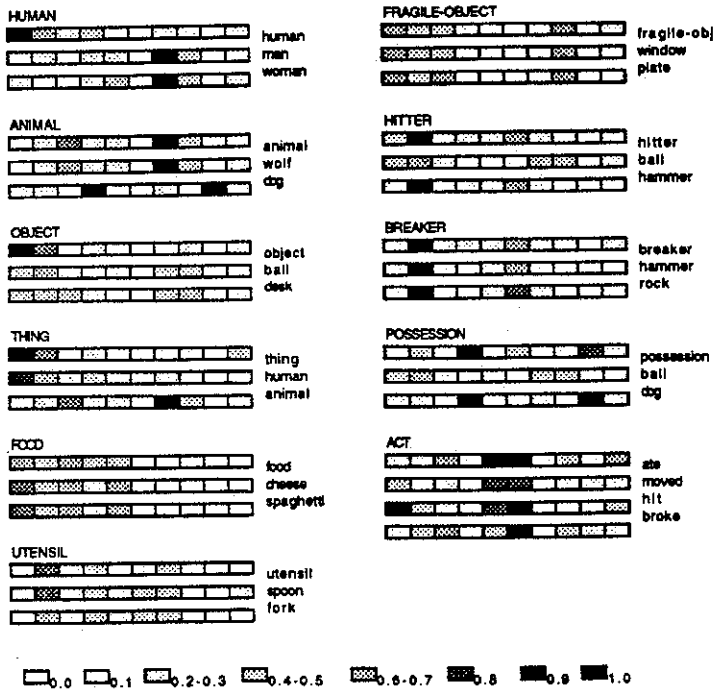

Figure 3: **Learned DSRs of concepts with their categories.** The experiment was done using momentum accelerated backpropagation [Rumelhart et al., 1986a, page 330]. Learning rate varied from 0.07 to 0.02; momentum factor varied from 0.5 to 0.9. The concepts and propositions were learned in 120 epochs; one epoch is 200 cycles of auto-associative backpropagation. The values range between 0.0-1.0, shown by the degree of box shading.

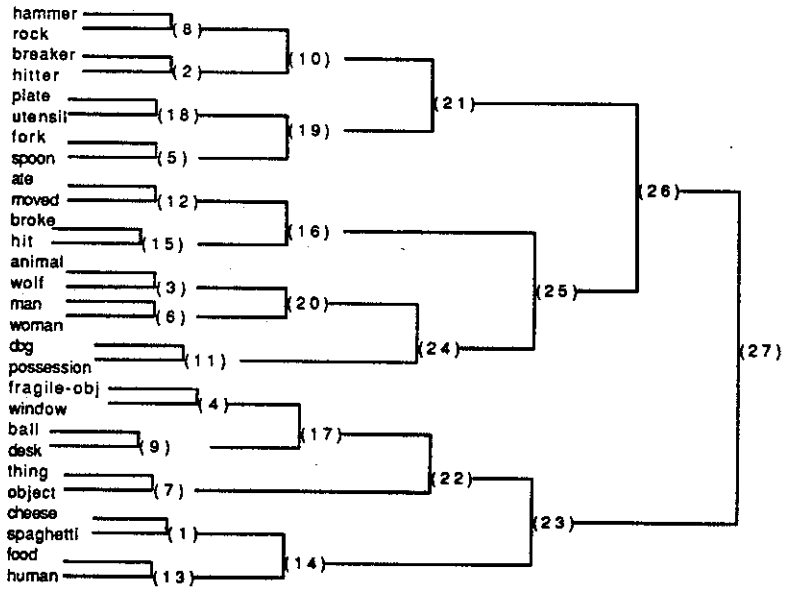

Figure 4: **Merge clustering the learned DSRs.** The numbers designate the time step. At each step, the clusters with the shortest average Euclidean distance were merged.

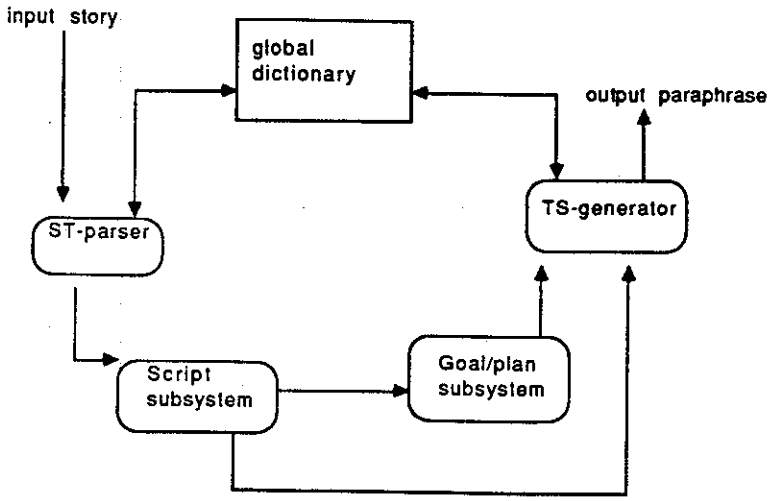

Figure 5: **DYNASTY top-level architecture.** The figure depicts the system during story processing performance phase. The uni/bi-directional arrows designate the information flow between modules. The GD contains word symbols and DSRs as common vocabularies.

Even if the two DSRs belong to the same category, the clustering steps are different according to the homogeneity of their usages. For example, the **cheese** and **spaghetti** are clustered at early time steps since they are mainly used as OBJECT, but **dog** and **possession** are clustered at later time steps because **dog** is also used as AGENT (in *animal* category) as well as as CO-OBJECT (in *possession* category). The somewhat non-intuitive clustering of **human** with **food** can be explained in the same way. The **human** also has multiple "usages", that is, **human** was used as both AGENT and OBJECT (Note **human** is also a concept filler for the *thing* category). But since **human** and **food** are not in the same category, they are clustered at a later step (step 13).

Interestingly, the representations of propositions also exhibit similarity structures, i.e. propositions involving similar case-roles and fillers have similar representations. These representations for the propositions can also be seen as higher-level representations for event structures. We postulate that this kind of event representation could be used in connectionist schema processing systems such as reported in [Dolan and Dyer, 1987; Lee et al., 1989]. DSRs show many similar properties to the Recursive Distributed Representations (RDR) [Pollack, 1990] with respect to recursiveness and structure encoding/decoding. But unlike RDRs, DSRs have word-level semantics in them so that they can be utilized not only in syntactic-level applications [Chalmers, 1990] but also in conceptual-level applications such as script-based story processing.

4 DYNASTY: Distributed Connectionist Story Paraphraser

DYNASTY [Lee et al., 1990] is a large-scale connectionist system that reads input stories and paraphrases the stories according to its knowledge of scripts and goals/plans. DYNASTY consists of several connectionist network modules, each communicating with the global dictionary (GD). Figure 5 shows top level DYNASTY architecture in the performance phase.

During the training phase, distributed semantic representations (DSR) for word-concepts are developed by the DSR-learner (discussed in the previous sections), and stored into the GD. During performance phase, DYNASTY reads the story word by word, and produces paraphrases of it.

Suppose the input story goes as follows:

```
John entered Chart-House.  John ate steak.  John left a tip.
```

The sentence-to-triple-parser (ST-parser) parses the input sentence into event triple forms such as:

[ev10 ACT **entered**], [ev10 AGENT **John**], [ev10 TO **Chart-House**].
[ev14 ACT **ate**], [ev14 AGENT **John**], [ev14 OBJECT **steak**].
[ev16 ACT **left**], [ev16 AGENT **John**], [ev16 OBJECT **tip**].

While the ST-parser is parsing the sentence, it accesses the GD to convert each word-concept symbol into a distributed pattern (DSR). GD is a symbol table which can be used to convert the symbols to the corresponding DSRs, and DSRs to their corresponding symbols. Next, event triples consisting of the distributed patterns are input to the script or goal/plan subsystem. In the event triples, the event numbers come from the training data and have no effect in story processing. If the script subsystem can select the appropriate script for the input story, it produces script-based paraphrases. If the script subsystem cannot find an appropriate script, it passes the event triples to the goal/plan subsystem where the story is paraphrased using goal/plan-knowledge. Since the example story is script-based, the script subsystem produces the script-based paraphrase in event triple forms such as:

[ev1 ACT **entered**], [ev1 AGENT **John**], [ev1 TO **Chart-House**].
[ev2 ACT **seated**], [ev2 AGENT **waiter**], [ev2 OBJECT **John**].
[ev3 ACT **brought**], [ev3 AGENT **waiter**], [ev3 OBJECT **menu**].
[ev4 ACT **read**], [ev4 AGENT **John**], [ev4 OBJECT **menu**].
[ev5 ACT **ordered**], [ev5 AGENT **John**], [ev5 OBJECT **steak**].
[ev6 ACT **ate**], [ev6 AGENT **John**], [ev6 OBJECT **steak**].
[ev7 ACT **paid**], [ev7 AGENT **John**], [ev7 OBJECT **bill**].
[ev8 ACT **left**], [ev8 AGENT **John**], [ev8 OBJECT **tip**].
[ev9 ACT **left-for**], [ev9 AGENT **John**], [ev9 FROM **Chart-House**], [ev9 TO **home**].

The triple-to-sentence generator (TS-generator) takes each subsystem's output and converts it to the surface forms such as:

John entered Chart-House. Waiter seated John. Waiter brought the menu. John read the menu. John ordered a steak. John ate the steak. John paid the bill. John left a tip. John left Chart-House for home.

The TS-generator accesses the GD network to convert the distributed pattern of each word-concept into its corresponding symbol while it is generating sentences from the event triples.

4.1 Script-processing subsystem

Figure 6 shows the script-processing subsystem with its intermediate representations.

The event-encoder takes the event triples and builds event representations (10-bit vector with continuous values). The sequence of event representations is fed to the script-recognizer, which selects an appropriate script pattern and its role-bindings. In the above example, it selects the restaurant script with customer bound to John, restaurant bound to Chart-House and food bound to steak. If it cannot select the proper script (the output of the script-recognizer is not clear), the event representations are input to the goal/plan-analysis subsystem to be processed. The selected script pattern goes to the backbone-generator, which produces the entire paraphrase as a sequence of event representations. The backbone event representations are decoded using event-encoder, which produces event triples (as shown in the previous section). The output of the script subsystem is input to the TS-generator, which generates the script-based paraphrase.

4.2 Goal/plan-analysis subsystem

The goal/plan-analysis subsystem has a similar architecture as the script-processing subsystem. When DYNASTY reads a goal-based story such as:

John was hungry. John picked up the restaurant-guide. John got into a car

it does not have an appropriate script trained for this sentence, and the event triples of this story go to the goal/plan subsystem. For this example, the ST-parser's output looks like the following (all word-concept symbols are replaced with their DSR patterns):

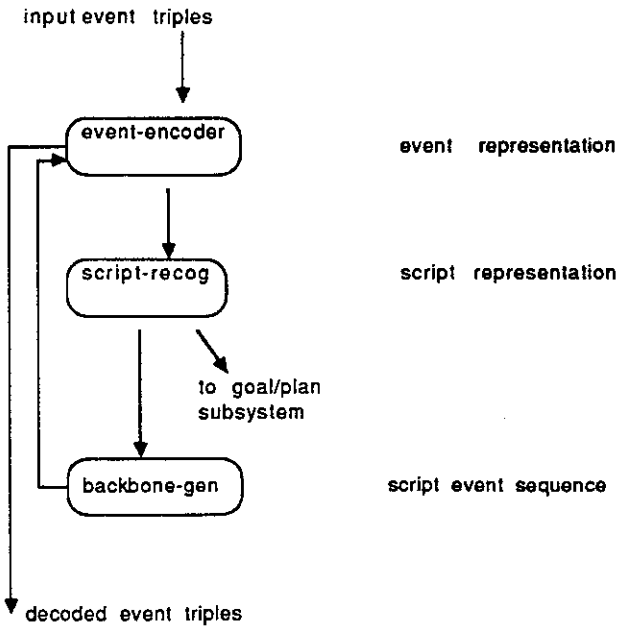

Figure 6: **Script processing subsystem architecture.** The arrows desigate data flow. Output of each module is shown beside the module.

[ev7 STATE **hungry**], [ev7 AGENT **John**].
[ev8 ACT **picked-up**], [ev8 AGENT **John**], [ev8 OBJECT **restaurant-guide**].
[ev9 ACT **got-into**], [ev9 AGENT **John**], [ev9 LOCATION **car**].

The goal/plan subsystem produces a goal/plan inference chain for each action of the planner as output paraphrases. Figure 7 shows the goal/plan analysis architecture.

The event-encoder produces the event representations (10-bit vector) from the event triples in the same manner as in the script subsystem. If the event is a state (first event)[1], then the GP-recognizer converts it to a goal representation, and makes the context goal of this story. If the event is an action, the GP-recognizer selects a proper goal/plan for this event using the first event as a context and passes the selected goal/plan representation to the GP-rel-associator. For example, for the first action (second event ev8 in above triples), the GP-recognizer selects the following plan (as a 10-bit plan representation) under the "ev7" context:

[p13 PLAN **pb-read**], [p13 AGENT **John**]. [p13 OBJECT **restaurant-guide**].

From this plan, the GP-rel-associator produces a goal/plan inference chain (under the satisfy-hunger context goal which is converted from the first event in the story) and decodes the goal/plan representations in the chain into the goal/plan triples such as:

[p13 PLAN **pb-read**], [p13 AGENT **John**]. [p13 OBJECT **restaurant-guide**].
[g2 GOAL **d-know**], [g2 AGENT **John**], [g2 OBJECT **restaurant-location**].
[p1 PLAN **pb-drive**], [p1 AGENT **John**], [p1 TO **restaurant**].
[g3 GOAL **d-cont**], [g3 AGENT **John**], [g3 OBJECT **food**].
[p2 PLAN **pb-eat**], [p2 AGENT **John**], [p1 OBJECT **food**].
[g1 GOAL **s-hunger**], [g1 AGENT **John**].

The chain production continues until GP-rel-associator produces an inferred (known) goal or plan. For the above example, the s-hunger goal was inferred from the first event so it is a known goal. From this output chain, the TS-generator produces the goal/plan-based paraphrase as an explanation for this event such as:

John will read the restaurant guide. Because John wants to know the restaurant-location. And then John will drive to the restaurant. Because John

[1] In the story, first event is always set up as a description of the planner's state from which we can extract the planner's current goal.

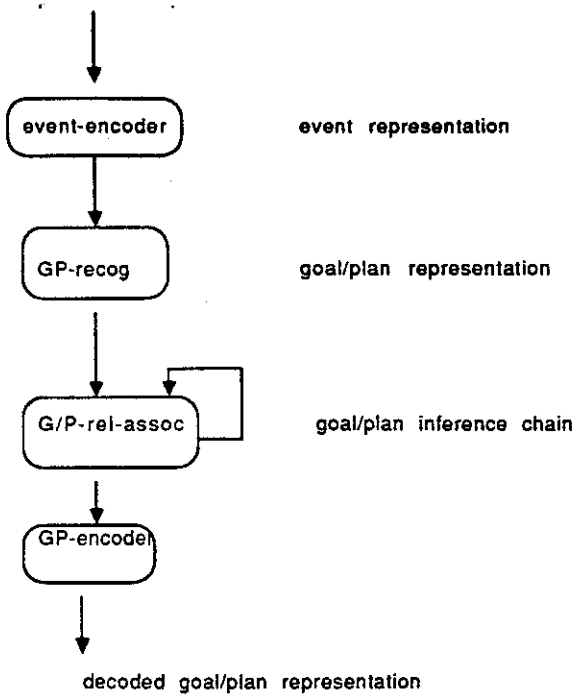

Figure 7: **Goal/plan-analysis subsystem architecture.**

```
wants to take control the food.  And then John will eat the food.  Because John
wants to satisfy-hunger.
```

The fixed word ("because" and "and then") is inserted automatically in front of each goal or plan. For the third event (ev9), the analysis process is the same except that DYNASTY considers all the goals and plans inferred during the past processes of this story as known goals and plans, and stops processing when it produces one of these known goals and plans.

4.3 DYNASTY training configuration

In the previous section, we discussed how DYNASTY processes script/goal-based stories. But processing is only possible after DYNASTY has been properly trained with appropriate training data. During training, DYNASTY extracts the statistical regularities in the data, which makes it possible to generalize into new stories. The different modules of DYNASTY are trained separately and in parallel. Training consists of two steps: (1) the DSRs are developed and stored in a global dictionary, and (2) each processing module is trained with the GD. Each module has its own training data, and is trained parallel using the DSRs in the GD as a common vocabulary.

5 Conclusion

We have proposed distributed semantic representations (DSRs) as an an adequate foundation for constructing and manipulating conceptual knowledge. DSRs are developed automatically in an XRAAM-based architecture. Our experiments indicate that DSRs have many desirable properties for high-level cognitive processing. We have shown that DSRs can serve as constituents of connectionist cognitive architectures by constructing an architecture that performs script/goal-based story understanding using recurrent connectionist modules (autoassociative or heteroassociative).

DYNASTY, a modular connectionist system for high-level inferencing can (1) automatically form distributed representations of word-concepts, events and goals/plans from the input sentences in the script/goal-based story understanding task, (2) generate complete script event sequences from fragmentary inputs, and (3) generate goal/plan inference chains from input actions. Moreover, the high-level representations (DSRs of concepts, events, scripts, and goals/plans) contain constituent structure that can be decoded and extracted, making the semantic content available for multiple tasks.

A lesson to be learned from the DYNASTY project is that many of the problems of the "one gigantic BP network"-approach in natural language understanding can be overcome with modular network architecture with suitable distributed representations and symbolic AI constraints.

References

Allen, R. B. (1988). Sequential connectionist networks for answering simple questions about a microworld. In *Proceedings of the Tenth Annual Cognitive Science Society Conference*, pages 489–495, Hillsdale, NJ. Lawrence Erlbaum Associates.

Chalmers, D. J. (1990). Syntatic transformations on distributed representations. *Connection Science.* (in press).

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 Annual International Conference on Neural Networks*, pages Vol. 2, 287–298. IEEE.

Dolan, C. P. and Smolensky, P. (1989). Implementing a connectionist production system using tensor products. In Touretzky, D. S., Hinton, G. E., and Sejnowski, T. J., editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 265–272, Los Altos, CA. Morgan Kaufmann Publishers, Inc.

Elman, J. L. (1988). Finding structure in time. Technical Report 8801, Center for Research in Language, University of California, San Diego.

Feldman, J. A. (1986). Neural representation of conceptual knowledge. Technical Report TR 189, Department of Computer Science, Univ. of Rochester, N.Y.

Fillmore, C. J. (1968). The case for case. In Bach, E. and Harms, R. T., editors, *Universals in Linguistic Theory*, pages 1–90. Holt, Rinehart and Winston, New York.

Hanson, S. J. and Kegl, J. (1987). Parsnip: A connectionist network that learns natural language grammar from exposure to natural language sentences. In *Proceedings of the Ninth Annual Cognitive Science Society Conference*, pages 106–119, Hillsdale, NJ. Lawrence Erlbaum Associates.

Hartigan, J. A. (1975). *Clustering Algorithms.* John Wiley and Sons, N.Y.

Hinton, G. E. (1986). Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Cognitive Science Society Conference*, pages 2–12, Hillsdale, NJ. Lawrence Erlbaum Associates.

Hinton, G. E. (1988). Representing part-whole hierarchies in connectionist networks. In *Proceedings of the Tenth Annual Cognitive Science Society Conference*, pages 48–54, Hillsdale, NJ. Lawrence Erlbaum Associates.

Hinton, G. E., McClelland, J. L., and Rumelhart, D. E. (1986). Distributed representations. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume I: Foundations*, pages 77–109. MIT Press, Cambridge, MA.

Jordan, M. I. (1986). Serial order: A parallel, distributed processing approach. Technical Report 8604, Institute for Cognitive Science, University of California, San Diego.

Lee, G., Flowers, M., and Dyer, M. G. (1989). A symbolic /connectionist script applier mechanism. In *Proceedings of the Eleventh Annual Cognitive Science Society Conference*, pages 714–721, Hillsdale, NJ. Cognitive Science Society, Lawrence Erlbaum Associates.

Lee, G., Flowers, M., and Dyer, M. G. (1990). Learning distributed representations of conceptual knowledge and their application to script-based story processing. *Connection Science.* (in press).

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 II: Psychological and Biological Models*, pages 272–326. MIT Press, Cambridge, MA.

Miikkulainen, R. and Dyer, M. G. (1988). Forming global representations with extended backpropagation. In *Proceedings of the IEEE Second Annual International Conference on Neural Networks*, pages Vol. 1, 285–292. IEEE.

Pollack, J. B. (1988). Recursive auto-associative memory: Devising compositional distributed representations. Technical Report MCCS-88-124, Computing Research Laboratory, New Mexico State University.

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence.* (in press).

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986a). Learning internal representations by error propagation. In Rumelhart, D. E. and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume I: Foundations*, pages 318–362. MIT Press, Cambridge, MA.

Rumelhart, D. E., McClelland, J. L., and the PDP Research Group (1986b). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* MIT Press, Cambridge, MA.

Schank, R. and Riesbeck, C. K., editors (1981). *Inside Computer Understanding.* Lawrence Erlbaum Associates, Hillsdale, NJ.

Smolensky, P. (1987). On variable binding and the representation of symbolic structures in connectionist systems. Technical Report CU-CS-355-87, Department of Computer Science and Institute of Cognitive Science, University of Colorado, Boulder.

Smolensky, P. (1988). On the proper treatment of connectionism. *The Behavioral and Brain Sciences*, 11(1):1–74.

St. John, M. F. and McClelland, J. L. (1989). Applying contextual constraints in sentence comprehension. In Touretzky, D. S., Hinton, G. E., and Sejnowski, T. J., editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 338–346, Los Altos, CA. Morgan Kaufmann Publishers, Inc.

Waltz, D. L. and Pollack, J. B. (1985). Massively parallel parsing: A strongly interactive model of natural language interpretation. *Cognitive Science*, (9):51–74.

20.3.12.