A Distributed Connectionist Representation for Concept Structures

David S. Touretzky and Shai Geva
Computer Science Department
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract. We describe a representation for frame-like concept structures in a neural network called DUCS. Slot names and slot fillers are diffuse patterns of activation spread over a collection of units. Our choice of a distributed representation gives rise to certain useful properties not shared by conventional frame systems. One of these is the ability to encode fine semantic distinctions as subtle variations on the canonical pattern for a slot. DUCS typically maintains several concepts simultaneously in its concept memory; it can retrieve a concept given one or more slots as cues. We show how Hinton's notion of a "reduced description" can be used to make one concept fill a slot in another.

1. Introduction

In a typical Lisp implementation of frames, a frame is a collection of named slots with fillers (Minsky, 1975; Winston & Horn, 1984). Names are atomic symbols, and fillers are either atoms or pointers to other frames. This paper considers what a connectionist version of frames might look like. We describe a representation for frame-like structures in a neural network called DUCS. Names and fillers are diffuse patterns of activation over a collection of units. Our choice of a distributed representation (Hinton *et al.*, 1986) for frames gives rise to certain useful properties that are not shared by conventional frame systems.

One thing a connectionist frame system should be able to do is retrieve a slot given an approximation of its name. For example, suppose the frame describing Fred the cockatoo had the following slots:

$$\left\{\begin{array}{rl} \text{BODY-COLOR:} & \text{PALE-PINK} \\ \text{BEAK:} & \text{GRAY-HOOKED-THING} \\ \text{CREST:} & \text{ORANGE-FEATHERED-THING} \\ \text{HABITAT:} & \text{JUNGLE} \\ \text{DIET:} & \text{SEEDS-AND-FRUIT} \end{array}\right\}$$

What does Fred's nose look like? Strictly speaking, birds don't have noses; they have beaks. If the activity patterns for NOSE and BEAK are similar, which they will be if we use

a microfeature-based representation for symbols, then a connectionist frame system could simultaneously retrieve the description GRAY-HOOKED-THING and correct the slot name to read BEAK instead of NOSE. Ordinary frame systems could not do this unless some rule of form "look for a beak if you can't find a nose" had been explicitly established prior to the query.

A second advantage of using a distributed representation for frames is that when slots are encoded as activity patterns, instead of being limited to a small, fixed set of slot names, the names form a continuous space. Subtle nuances of meaning of the frame as a whole can be encoded as variations on the activity patterns of its slots. This is useful in the case role representation of sentences. For example, in "John sold the statue to Mary," Mary plays the role of recipient. In "John mailed the package to Boston," Boston's role would be destination. But in "John threw the ball to Mary" the role of Mary is both destination (the ball is thrown in her direction and is expected to make contact with her) and recipient (John's intention is that the ball come into her possession and be under her control.) In a system based on distributed representations, the pattern representing Mary's role could be a combination of destination and recipient, sharing microfeatures of both. Finer shadings of role names are also possible. We propose that in a connectionist version of case grammar, each combination of a verb, some case roles, and the fillers of those roles would generate slightly different role name patterns based on subtle nuances of the meaning of the sentence as a whole.

This idea was anticipated in McClelland and Kawamoto's PDP model of case role assignment (McClelland & Kawamoto, 1986). Their model provides four case slots called agent, patient, instrument, and modifier; a representation of the verb is conjunctively encoded with each slot filler. Although the names of the four slots are fixed, the fillers undergo variations from their canonical, surface forms according to the context in which they appear. For example, while there is only one pattern for representing the verb "move" in the input layer, in the case role layer the representation of "move" will be different when it has an animate agent that implicitly moves itself ("the cat moved") versus an agent that moves other things ("the boy moved the cat") versus an inanimate subject ("the rock moved") which is interpreted as a patient with the agent left unspecified.

In the following sections we describe the architecture of DUCS. The name stands for Dynamically Updatable Concept Structures. We will use the word concept rather than frame in the remainder of the paper in order to distinguish DUCS' structures from the ones used in Lisp-based reasoners. We will discuss two problems peculiar to connectionist systems. One is the problem of getting a concept to fill a slot in another concept. The other is the problem of getting concepts not recently accessed to automatically fade from working memory as new ones are created, so that the memory capacity is not exceeded.

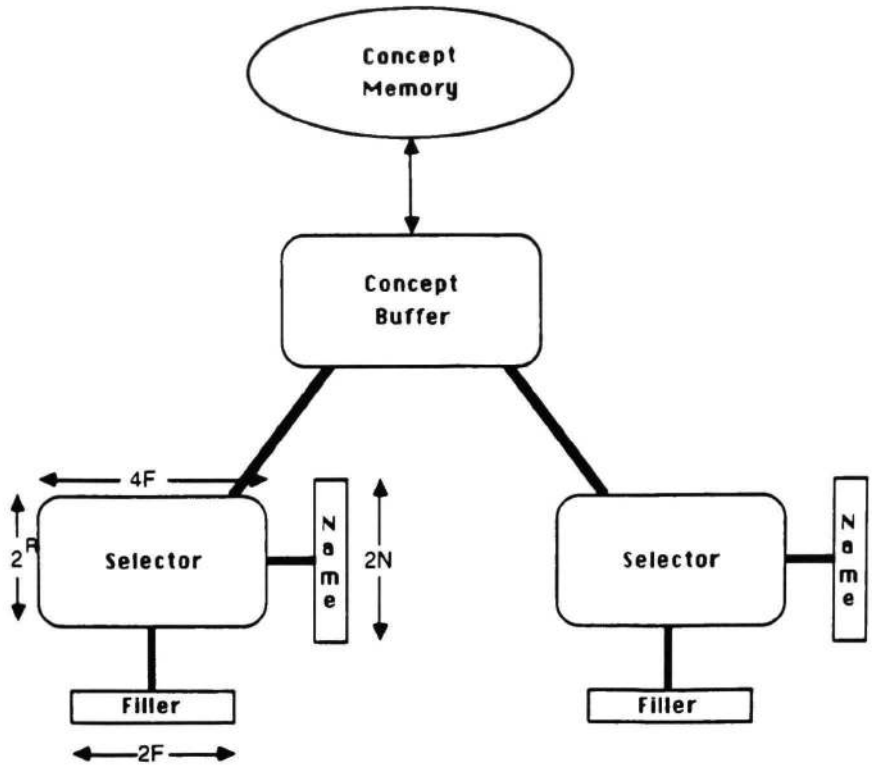

Figure 1: The DUCS Architecture.

2. The DUCS Architecture

Both slot names and slot fillers in DUCS employ distributed representations, meaning they exist as diffuse patterns of activity over a collection of units. Each slot name pattern determines a mapping of the associated slot filler pattern into an array called the concept buffer. The patterns that various slots generate in the buffer are superimposed to derive a pattern for the entire concept. See Figure 1.

DUCS is a two-level architecture. At the slot level, it retrieves individual slots by name, and can add, change, or delete slots by modifying the activity pattern in the concept buffer. Through the use of multiple slot mapping assemblies, each consisting of a slot name group, a slot filler group, and a selector group, several slots can be created or modified simultaneously. Thus it is possible to create a complex concept in a single operation as long as the number of slots does not exceed the available mapping hardware. Slots can also be loaded into the buffer sequentially.

At the concept level, DUCS manipulates entire concepts at a time rather than individual slots. Concepts are added to or deleted from an auto-associative concept memory via the concept buffer. DUCS retrieves a concept from concept memory using one or more slots as cues, in the following way. First the cues are loaded into slot mapping assemblies, where they generate a partial activity pattern in the concept buffer. Then the concept buffer

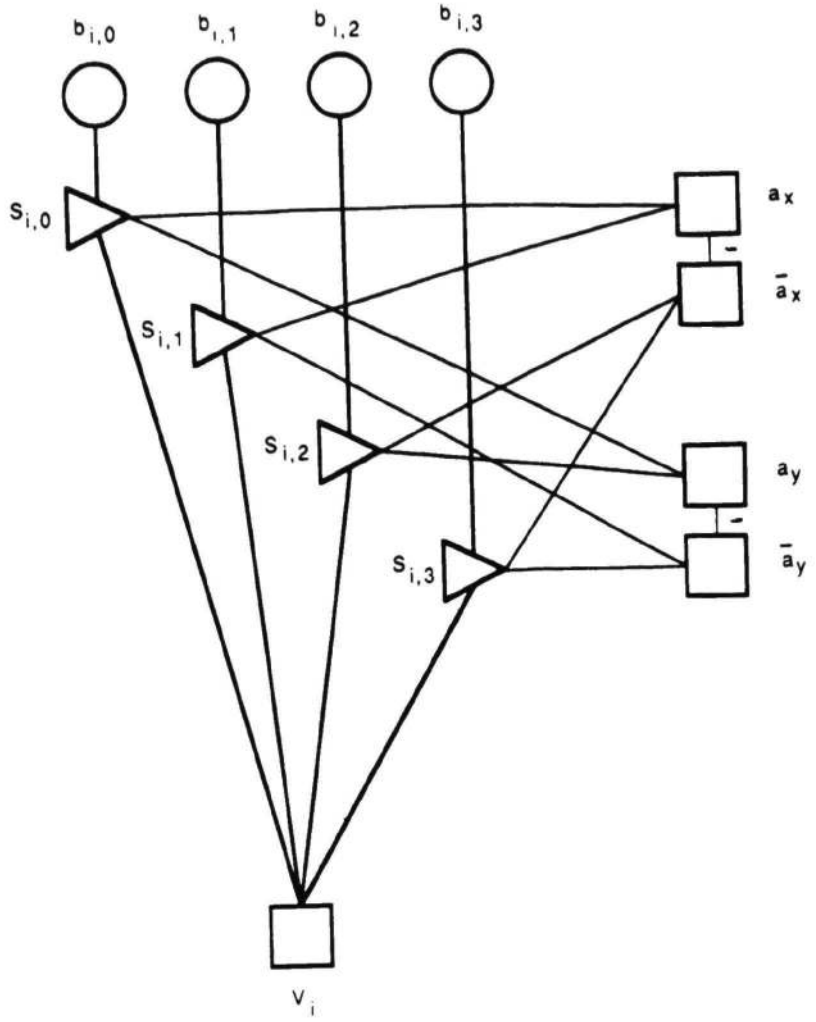

Figure 2: Mapping the slot filler bit v_i into one of 2^R concept memory bits based on an R-bit subset of the slot name. Mutually inhibitory connections among the 2^R selector units have been omitted for clarity.

supplies input to the concept memory, allowing it to complete the pattern and cause the remaining slots of the concept to materialize in the buffer.

2.1. The Slot Level

Slot names and slot fillers are N and F bit binary feature vectors, respectively, appended to their logical complements. That is, a slot name $\vec{a}$ is a $2N$-bit vector such that $a_i = \overline{a}_{(i+N)}$, $1 \leq i \leq N$. Similarly, a slot filler $\vec{v}$ is a $2F$-bit vector where $v_i = \overline{v}_{(i+F)}$, $1 \leq i \leq F$. The concept buffer and each selector group are $4F \times 2^R$ arrays, where R is a parameter between 0 and N.

Storing the filler pattern $\vec{v}$ in the slot named $\vec{a}$ generates a $4F \times 2^R$ pattern over the selector array $s_{i,j}$. Each bit v_i is copied into one of the 2^R locations in column i of the array, and independently, into one of the 2^R locations in column $i + 2F$. The location j

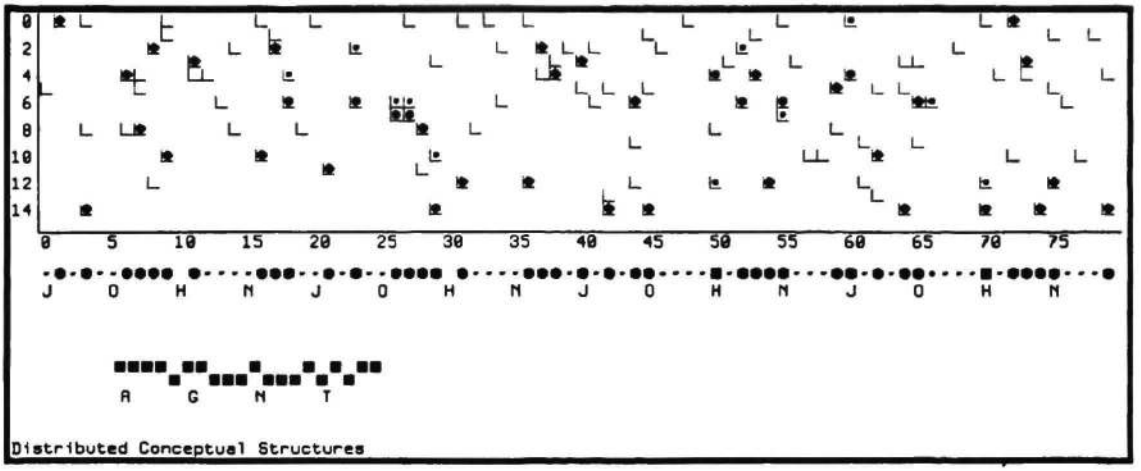

Figure 3: The state of the network part way through a retrieval of the AGENT slot.

within the column is determined by an R-bit subset of the slot name; a different subset is randomly associated with each column. After the selector group has stabilized, the active units $s_{i,j}$ are allowed to excite the corresponding units $b_{i,j}$ in the concept buffer, thereby superimposing the pattern for this slot onto the previous contents of the buffer.

Figure 2 shows the wiring for mapping the value of a single slot filler bit v_i into some $s_{i,j}$ in column i. Selector units within a column form mutually-inhibitory winner-take-all networks, so that in a stable state at most one selector per column will be active. The chosen unit will be the one with the most activation, *i.e.*, the one with the most of its R input lines from the slot name group active. In the figure, $R = 2$, so the position j is determined by two slot name bits. Note that v_i will also be copied into some position k in column $i+2F$, where k is determined by a different pair of bits randomly chosen from the slot name. Also, v_{i+F}, which is $\overline{v}_i$, will be copied into columns $i+F$ and $i+3F$ in positions determined by two other pairs of slot name bits. Thus the pattern developed in the selector group consists of two copies of each filler bit v_i and two copies of its complement, with each copy deposited in one of the 2^R positions in that column determined by the slot name.

The units in the slot name, slot filler, and selector groups are continuous-valued non-linear units with outputs restricted to the unit interval; all connections are symmetric. This is commonly known as a Hopfield and Tank model (Hopfield & Tank, 1985). Retrievals are accomplished by clamping the concept buffer and slot name space and setting a low gain value for selector and slot filler units. The gain then rises fairly quickly, and part way through the slot name group is unclamped. At high gain the network settles into a stable state representing the slot filler and (possibly corrected) slot name extracted from the concept buffer. Figure 3 shows the state of the network at a medium gain setting. Here, slot names and fillers are ASCII strings (using a five bit character code) rather than

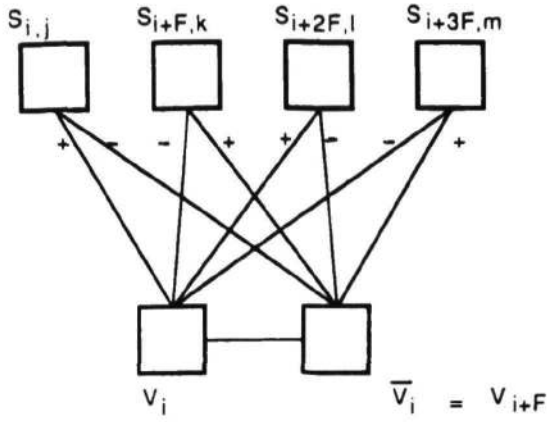

Figure 4: Error correction circuitry to exploit the redundant storage of slot fillers.

vectors of microfeatures, in order to make the model's operation more transparent. The bottom group of units represents the slot name "AGNT". The vector above that shows four copies of the slot filler "JOHN", two of which are logically inverted. The array at the top of the figure depicts the activity in the concept buffer (L-shaped symbols) and selector group (solid circles.) Three slots have been stored in the buffer; therefore each column of the array has between zero and three L-shapes, indicating active concept buffer units $b_{i,j}$. The size of the circles indicates the output level of the corresponding selector units $s_{i,j}$. Most columns have only one large circle, but some have two, indicating a pair of selectors in competition. At high gain all units will be either fully on or fully off, and there will be at most one active selector per column.

Selector units function as skeleton filters, so-called because only a skeleton subset of the units are enabled at any one time. Hinton (1981) used skeleton filters in a connectionist implementation of a semantic network. Sejnowski (1981) presents arguments for the existence of skeleton filters in the brain.

The redundant storage of fillers helps correct errors that may occur when several slots are superimposed in the concept buffer. Two slots can potentially interfere at column i when the R-bit subset of the slot name group examined by that column yields the same j value for both slots. If the first slot has a 1 in bit v_i and the second slot has a 0 (which implies the reverse situation in bit v_{i+F}), $s_{i,j}$ will be set to 1. But each filler bit is stored several times using a different R-subset each time, and the two slots are unlikely to overlap in every copy. Figure 4 shows the error correction circuitry used during slot retrieval to derive bits v_i and v_{i+F} from $s_{i,j}$, $s_{i+F,k}$, $s_{i+2F,l}$, and $s_{i+3F,m}$ via a majority voting scheme, where j, k, l, and m are determined by different R-subsets. The error correction imposes a constraint on the filler pattern that it have F bits on, and that $v_i = \overline{v}_{i+F}$, $1 \leq i \leq F$.

The error correction scheme is also important for associative retrieval. Suppose the

network tries to retrieve a slot with a few bits of the slot name in error. Most of the $4F$ different R-subsets of the slot name will pick bits that are correct, so most filler bits will be mapped to the correct positions in their respective columns. Those columns that reference incorrect slot name bits will not be mapped properly. The error correction circuitry puts pressure on the slot filler group to settle into a pattern that meets the above mentioned constraints, and this in turn puts pressure on the selector units. If enough pressure is applied, the selector units can force the slot name units to change. This is how the network can change NOSE to BEAK during retrieval from a bird concept, assuming that the two symbols have similar activity patterns and a valid filler exists in the BEAK slot.

The parameter R determines the number of slots the concept buffer can hold. Since each filler bit maps into one of 2^R positions in its column, increasing R increases the sparseness of the concept buffer and reduces the chance that slots will interfere. Another way to increase the capacity of the concept buffer would be to to widen the array from $4F$ to, say, $6F$ units, to provide improved error correction.

2.2. The Concept Level

DUCS can memorize or retrieve entire concepts in a single operation. The activity pattern for a concept is a bit vector of length $4F \times 2^R$. The concept memory, shown in Figure 1, is a $(4F \times 2^R)$ by $(4F \times 2^R)$ matrix forming a Willshaw-style auto-associative net (Willshaw, 1981). For every pair of active concept buffer units $b_{i,j}$ and $b_{i*,j*}$ there is a concept memory unit. To store a pattern in concept memory it suffices to turn on each concept memory unit whose associated pair of concept buffer units is active.

To retrieve a concept, some subset of the pattern, generated by whatever cues were supplied, is fed into the auto-associative net. The retrieval is based on the assumption that the retrieval cues are correct. It yields a superposition of all patterns for concepts stored in the concept memory that match the given cues. For example, if two concepts with John as agent had been stored, both would be retrieved from the single cue AGENT=JOHN. A more detailed cue would be required to restrict the retrieval to a single concept. On the other hand, if no bits in addition to the original cue are retrieved, then the cue is a full specification of the desired concept, or else no such concept is stored in the memory. The network can detect whether a retrieval was successful by checking whether all pairs of bits in the concept buffer have their associated concept memory unit active. If this is the case, then the cue supplied was a valid one and only a single concept was retrieved.

3. Naming and Reduced Descriptions

In Lisp it's easy to make one structure point to another. One way to achieve a similar effect in connectionist models is to use a technique called reduced descriptions (Hinton,

1987.) For example, to represent "Bill knows that John kissed Mary" we first create and store the concept "John kissed Mary." Then we derive a small pattern for this concept, the reduced description, to fill the patient role in the concept "Bill knows that x," as shown:

$$\left\{\begin{array}{rl} \text{AGENT:} & \text{JOHN} \\ \text{VERB:} & \text{KISS} \\ \text{PATIENT:} & \text{MARY} \end{array}\right\} \qquad \left\{\begin{array}{rl} \text{AGENT:} & \text{BILL} \\ \text{VERB:} & \text{KNOW} \\ \text{PATIENT:} & \text{JhnKssMry} \end{array}\right\}$$

In order for this technique to work, the network must be able to retrieve the full pattern for a concept given its reduced description. In DUCS we obtain the reduced description simply by taking an F-bit slice out of the concept buffer. The exact choice of slice is unimportant; currently we use $b_{0,0}$ through $b_{F,0}$. To "follow the pointer" in the patient slot to get to the full description of what Bill knows, the F-bit filler pattern is clamped into bits $b_{0,0}$ through $b_{F,0}$ of the concept buffer, and this serves as the cue for the associative network to retrieve the rest of the concept pattern. Touretzky (1986) describes another version of pointer following in a connectionist network which does not use reduced descriptions.

4. Forgetting

The concept memory has a limited capacity. If a reasoner continually generates and stores new concepts, the memory could fill up, making further processing impossible. We view DUCS as a short term working memory for concept structures. In order to prevent its memory capacity from being exceeded, we implemented a forgetting mechanism by which concepts not recently accessed can fade and be displaced by newly stored ones.

Each unit in the concept memory has two parameters: an internal integer activation value in the interval $[0, c]$, and a binary output value that is 1 whenever the activation value is positive. To memorize a concept pattern, the first step is to decrement the activity levels of all concept memory units by one. Any unit whose activity has decayed to zero turns off. Then, for each pair of active units $b_{i,j}$ and $b_{i*,j*}$ in the concept buffer, the corresponding concept memory unit is given an initial activity level of c. With this protocol, concept memory will hold at most c distinct concepts at a time, provided that concepts are reinforced in concept memory only after non-ambiguous retrieval.

For auto-associative retrieval, concept memory units are treated as binary state units, with any non-zero activation value indicating a 1 state. Whenever a concept is retrieved into the concept buffer, it is immediately re-stored into concept memory. This refreshes the concept memory units by setting their activation levels back to c. Concepts which have not been fetched from concept memory for a while eventually decay due to lack of refresh. Non-active memory units are never reinforced, even if their associated pair of concept buffer units is active, to prevent undesired merging of multiple concepts retrieved simultaneously due to ambiguous cues.

5. Discussion

There are many architectures for building associative memories (Hinton & Anderson, 1981; Baum *et al.*, 1987). One of the unique features of DUCS is its two level structure: concepts can be recalled from concept memory into the concept buffer using slots as cues, and slots can be recalled from the concept buffer using their names as cues. The fact that slots are represented as activity patterns rather than as weights makes this possible.

Hinton's reduced description idea has great potential which we have only begun to explore. He suggests that the reduced description pattern should be meaningful (*i.e.*, interpretable) in its own right. Our current version contains too few bits to meet this criterion. An enhanced version might contain information about the type of the concept and a summary description of its slots. This would make it possible to make gross inferences about concepts (*e.g.*, that Bill knows a fact about some male person kissing some female person) without having to expand each reduced description beforehand. An ideal reduced description mechanism, rather than just taking a fixed slice out of a concept, would detect and focus on relevant features to evolve the most meaningful set of reduced descriptions possible in a given domain. We don't yet know how this might be accomplished.

Acknowledgements

Mark Derthick's μKLONE system (Derthick 1987; Touretzky & Derthick 1987), still under development, was a major source of architectural ideas for DUCS. We thank Mark Derthick and Geoffrey Hinton for frequently illuminating discussions. This work was supported in part by National Science Foundation grant IST-8516330, and by the Office of Naval Research under contract number N00014-86-K-0678.

References

[1] Baum, E. B., Moody, J., and Wilczek, F. (1987) Internal representations for associative memory. Manuscript. Institute for Theoretical Physics, University of California at Santa Barbara.

[2] Derthick, M. A. (1987) Factual and counterfactual reasoning by constructing plausible models. Manuscript. Computer Science Department, Carnegie Mellon University.

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

[4] Hinton, G. E. (1987) Representing part-whole hierarchies in connectionist networks. Manuscript. Computer Science Department, Carnegie Mellon University.

[5] Hinton, G. E., McClelland, J. L, and Rumelhart, D. E. (1986) Distributed representations. In D. E. Rumelhart and J. L. McClelland (eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. Cambridge, MA: The MIT Press.

[6] Hopfield, J. J. and Tank, D. (1985) "Neural" computation of decisions in optimization problems. *Biological Cybernetics, 52*, 141-152.

[7] McClelland, J. L. and Kawamato, A. H. (1986) Mechanisms of sentence processing: assigning roles to constituents. In J. L. McClelland and D. E. Rumelhart (eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2. Cambridge, MA: The MIT Press.

[8] Minsky, M. L. (1975) A framework for representing knowledge. In P. H. Winston (ed.), *The Psychology of Computer Vision*. New York: McGraw-Hill.

[9] Sejnowski, T. J. (1981) Skeleton filters in the brain. In Hinton, G. E. and Anderson, J. A (eds.), *Parallel Models of Associative Memory*. Hillsdale, NJ: Earlbaum.

[10] Touretzky, D. S. (1986) BoltzCONS: reconciling connectionism with the recursive nature of stacks and trees. *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, Amherst, MA.

[11] Touretzky, D. S. and Derthick, M. A. (1987) Symbol processing in connectionist networks: five properties and two architectures. *Proceedings of IEEE Spring COMPCON87*, San Francisco.

[12] Willshaw, D. (1981) Holography, associative memory, and inductive generalization. In Hinton, G. E. and Anderson, J. A (eds.), *Parallel Models of Associative Memory*. Hillsdale, NJ: Earlbaum.

[13] Winston, P. H. and Horn, B. K. P. (1984) *Lisp*, second edition. Boston: Addison-Wesley.