Recursive Auto-Associative Memory: Devising Compositional Distributed Representations

Jordan Pollack

Computing Research Laboratory
New Mexico State University

INTRODUCTION

A major outstanding problem for connectionist models is the representation of variable-sized recursive and sequential data structures, such as trees and stacks, in fixed-resource systems. Such representational schemes are crucial to efforts in modeling high-level cognitive faculties, such as Natural Language processing. Pure connectionism has thus far generated somewhat unsatisfying systems in this domain, for example, which parse fixed length sentences (Cottrell, 1985; Fanty 1985; Selman, 1985; Hanson & Kegl, 1987), or flat ones (McClelland & Kawamoto, 1986).[1]

Thus, one of the main attacks on connectionism has been on the inadequacy of its representations, especially on their lack of compositionality (Fodor & Pylyshyn, 1988).

However, some design work has been done on general-purpose distributed representations with limited capacity for sequential or recursive structures. For example, Touretzky has developed a coarse-coded memory system and used it both in a production system (Touretzky & Hinton, 1985) and in two other symbolic processes (Touretzky, 1986ab). In the past-tense model, Rumelhart and McClelland (1986) developed an implicitly sequential representation, where a pattern of well-formed overlapping triples could be interpreted as a sequence.

Although both representations were successful for their prescribed tasks, there remain some problems.

- First, a large amount of human effort was involved in the design, compression and tuning of these representations.
- Second, both require expensive and complex access mechanisms, such as pullout networks (Mozer, 1984) or clause-spaces (Touretzky & Hinton, 1985).
- Third, they can only encode structures composed of a fixed tiny set of representational elements, (i.e. like triples of 25 tokens), and can only represent a small number of these element-structures before spurious elements are introduced[2]. These representational spaces are, figuratively speaking, like a "prairie" covered in short bushes of only a few species.
- Finally, they utilize only binary codes over a large set of units.

The compositional distributed representations devised by the technique to be described below demonstrate somewhat opposing, and, I believe, better properties:

- Less human work in design by letting a machine do the work,
- Simple and deterministic access mechanisms,
- A more flexible notion of capacity in a "tropical" representational space: a potentially very large number of primitive species, which combine into tall, but sparse structures.
- Finally, the utilization of analog encodings.

The rest of this paper is organized as follows. First, I describe the strategy for learning to represent stacks and trees, which involves the co-evolution of the training environment along with the access mechanisms and distributed representations. Second, I allude to several experiments using this strategy, and provide the details of an experiment in developing representations for binary syntactic trees. And, finally, some crucial issues are discussed.

RECURSIVE AUTO-ASSOCIATIVE MEMORY

Learning To Be A Stack

Consider a variable-depth stack of L-bit items. For a particular application, both the set of items (i.e. a subset of the 2^L patterns) and the order in which they are pushed and popped are much more constrained than, say, all possible sequences of N such patterns, of which there are 2^{LN}. Given this fact, it should be possible to build a stack with less than LN units (as in a shift-register approach) but more than L units with less than N bits of analog resolution, as in an approach using fractional encodings such as the one I used in the construction of a "neuring machine" (Pollack, 1987a).

[1] It is possible to get around some of these problems by resorting to hybrid modeling, e.g. (Waltz & Pollack, 1985).

[2] Rosenfeld and Touretzky (1987) provide a nice analysis of coarse-coded symbol memories.

The problem is finding such a stack for a particular application.

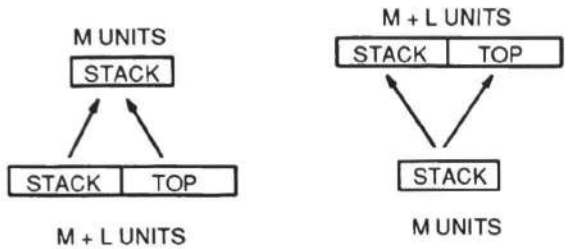

Figure 1.

Proposed inverse stack mechanisms in single-layered feedforward networks.

Consider representing a stack in a activity vector of M bounded analog values, where $M>L$. Pushing a L-bit vector onto the stack is essentially a function that would have $L+M$ inputs, for the new item to push plus the current value of the stack, and M outputs, for the new value of the stack. Popping the stack is a function that would have M input units, for the current value of the stack, and $L+M$ output units, for the top item plus the representation for the remaining stack. Potential mechanisms in the form of single-layered networks are shown in figure 1. The operation performed by a single layer is a vector-by-matrix multiplication and then a non-linear scaling of the output vector to between 0 and 1 by a logistic function.

All we need for a stack mechanism then, are these two functions plus a distinguished M-vector of numbers, ε, the *empty* vector. To push elements onto the stack, simply encode the element plus the current stack; to pop the stack, decode the stack into the top element and the former stack. Note that this is a recursive definition, where previously encoded stacks are used in further encodings. The problem is that it is not at all clear how to design these functions, which involve some magical way to *recursively* encode $L+M$ numbers into M numbers while preserving enough information to consistently decode the $L+M$ numbers back.

One clue for how to do this comes from the Encoder Problem (Ackley, Hinton, & Sejnowski, 1985), where a sparse set of fixed-width patterns are encoded into a set of patterns of smaller width. Back-propagation has been quite successful at this problem, when used in an unsupervised autoassociative mode on a three layer network. Rumelhart, Hinton, & Williams (1986) only demonstrated an 8-3-8 encoder,[3] but Cottrell, Munro, & Zipser (1987) demonstrated a 64-16-64 encoder and Hanson & Kegl (1987) used a 270-45-270 network.

These encoder networks are not directly applicable to the stack mechanism, because the compressed representations are never further processed. But they can be.

Consider a set of training examples for a stack as snapshots of the deepest states some procedure using that stack creates. For example, if the procedure performed the following stack operations generating the corresponding stack states:

operation	stack state
PUSH A	(A)
PUSH B	(B A)
PUSH C	(C B A)
POP	(B A)
PUSH D	(D B A)
PUSH C	(C D B A)
POP	(D B A)
POP	(B A)
POP	(A)
PUSH D	(D A)
POP	(A)
POP	()

Then the deepest states created are (C B A), (C D B A), and (D A), since all the other stack states are "substacks" of these three.

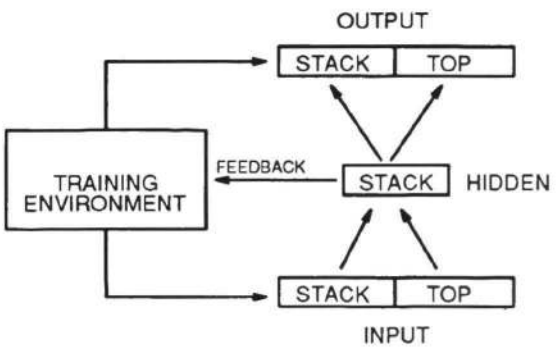

Figure 2.

Recursive Autoassociative Memory. The memory develops compositional distributed representations as the outputs of the hidden layer. The developing representations are fed back into the training environment, which therefore evolves with the weights in the network.

Consider simultaneously training the push and pop mechanisms from figure 1. Taken together, they form an encoder network as shown in Figure 2, with $L+M$ input units, M hidden units, and $L+M$ output units. If the symbols

[3] The three numbers correspond to the number of units in each layer of a 3-layer feed-forward network. I will not describe back-propagation here, assuming that the reader is familiar with the technique. Any programmed implementation of it can be simply modified to do recursive auto-association. These modifications, which involve using two error tolerances, and a mechanism to save, restore, and copy the output values of the hidden layer into the input layer, will be made obvious through the discussion.

above, A through D, represent L-bit vectors, and ε is a distinguished vector of size M, then a successful application (to be defined below) of back-propagation over this network with the following set of training examples can develop the mechanisms for this stack. The roman letters indicate particular L-bit patterns, $R_i(t)$ represents the M values of the hidden units for a particular example during a particular training epoch, and + **is used to indicate concatenation rather than addition**.

input		hidden		output
$A+\varepsilon$	$\rightarrow$	$R_A(t)$	$\rightarrow$	$A'+\varepsilon'$
$B+R_A(t)$	$\rightarrow$	$R_{BA}(t)$	$\rightarrow$	$B'+R_A(t)'$
$C+R_{BA}(t)$	$\rightarrow$	$R_{CBA}(t)$	$\rightarrow$	$C'+R_{BA}(t)'$
$A+\varepsilon$	$\rightarrow$	$R_A(t)$	$\rightarrow$	$A'+\varepsilon'$
$B+R_A(t)$	$\rightarrow$	$R_{BA}(t)$	$\rightarrow$	$B'+R_A(t)'$
$D+R_{BA}(t)$	$\rightarrow$	$R_{DBA}(t)$	$\rightarrow$	$D'+R_{BA}(t)'$
$C+R_{DBA}(t)$	$\rightarrow$	$R_{CDBA}(t)$	$\rightarrow$	$C'+R_{DBA}(t)'$
$A+\varepsilon$	$\rightarrow$	$R_A(t)$	$\rightarrow$	$A'+\varepsilon'$
$D+R_A(t)$	$\rightarrow$	$R_{DA}(t)$	$\rightarrow$	$D'+R_A(t)'$

There are several nonobvious things going on here.

First of all, the (initially random) values of the hidden units, $R_i(t)$, are part of the training environment. As the weights in the system evolve, so does the training environment. The stability and convergence of the network is thus sensitive to the learning rate. It must be set low enough that the change in the hidden representations does not invalidate the decreasing error granted by the change in weights.

In lieu of a formal proof of convergence, I simply argue inductively that the terminal training patterns are constant, so that the patterns of depth 1 become constant as learning proceeds; as they become stable, the patterns of depth 2 start to stabilize, and so on. This paper focuses on the strategy for finding these representations, so while stability and convergence in learning is an important problem, here it is only a secondary issue. As will be discussed later, it may be possible to solve auto-association problems directly through algebraic means.

The second nonintuitive point is that the network is not really being trained as a stack! Considering that the input-to-hidden function performs a PUSH, and the hidden-to-output function function a POP, the training regime involves multiple PUSHes, but only single POPs.

Enter the notion of *successful training*, by which I mean the usual termination condition for back-propagation: For each example, and for each unit, the absolute value of the difference between desired and actual output is less than some tolerance.

For the encoder and decoder to really work, however, the tolerance on the developed representations has to be quite sharp. After pushing A, the stack is represented by R_A. After pushing B, the stack is represented by R_{BA}. Popping R_{BA} returns B′ and R_A′. In order to successfully pop R_A′ to get A′ and ε', R_A′ must be very similar to R_A.

In running the simulations, then, I use two different tolerances, a terminal tolerance, τ, for the input/output bit-vectors, and another non-terminal tolerance, ν for the developing hidden representations. In the experiments described below, I use τ=0.2 and ν=0.05.

The third conceptual problem is that the two parts of the RAAM really define separate mechanisms, a PUSH encoder and a POP decoder, which will be embedded in different parts of an application network with some simpler sort of memory and control logic. That these mechanisms are tightly coupled in training does not mean that they always form a single network.

I have run several experiments in learning distributed representations for sequences whose details are omitted for lack of space. One 21-12-21 RAAM learned to be a stack for a network using a recursive subgoal strategy to solve the Towers-of-Hanoi Puzzle; Another 15-10-15 RAAM learned to represent sequences of letters (encoded as 5 bit numbers) in common words; and a third experiment on a 4-3-4 RAAM trained on all eight depth-3 patterns of a single bit stack developed into a shift register.

Learning to Represent Trees

Considering that a stack is really a right-branching binary tree with a distinguished empty symbol, it should be obvious that this mechanism can also be adapted for dealing with other *fixed valence* trees. For a training set consisting of unlabeled binary trees where the terminals are K-bit binary vectors, a three-layer network with $2K$ input units, K hidden units, and $2K$ output units can be used to develop representations for a set of such trees. The input-to-hidden function encodes two trees into a new, higher-level, tree, while the hidden to output function decodes a tree into two subtrees.

Consider the tree, ((D (A N))(V (P (D N))), as one member of a training set of such trees. If the aforementioned network is successfully trained with the following patterns (among other such patterns in the training environment), the resultant encoder and decoder networks can reliably form representations for these binary trees.

input	hidden	output
A+N	$\rightarrow R_{AN}(t)$	$\rightarrow$ A'+N'
D+$R_{AN}(t)$	$\rightarrow R_{DAN}(t)$	$\rightarrow$ D'+$R_{AN}(t)'$
D+N	$\rightarrow R_{DN}(t)$	$\rightarrow$ D'+N'
P+$R_{DN}(t)$	$\rightarrow R_{PDN}(t)$	$\rightarrow$ P'+$R_{DN}(t)'$
V+$R_{PDN}(t)$	$\rightarrow R_{VPDN}(t)$	$\rightarrow$ V'+$R_{PDN}(t)'$
$R_{DAN}(t)$+$R_{VPDN}(t)$	$\rightarrow R_{DANVPDN}(t)$	$\rightarrow R_{DAN}(t)'$+$R_{VPDN}(t)'$

DETAILS OF AN EXPERIMENT

In fact, the above example was part of the first experiment I ran on learning to represent trees. Consider a simple grammar, where every rule expansion has exactly two components:

S -> NP VP | NP V
NP -> D AP | D N | NP PP
PP -> P NP
VP -> V NP | V PP
AP -> A AP | A N

Given a set of strings in the language defined by this grammar, any old parser is capable of returning bracketed binary trees which will make up our training set. I made up such a set of strings, and used a parser to get the following set of trees:

(D (A (A (A N))))
((D N)(P (D N)))
(V (D N))
(P (D (A N)))
((D N) V)
((D N) (V (D (A N)))))
((D (A N)) (V (P (D N))))

Each terminal (D A N V & P) was represented as a 1-bit-in-5 code padded with 5 zeros. A 20-10-20 RAAM devised the representations shown in figure 3.

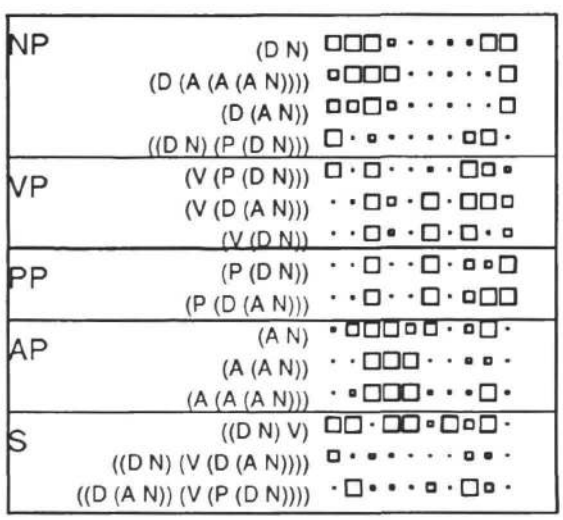

Figure 3.

Representations of all the binary trees in the training set, devised by a 20-10-20 RAAM, manually clustered by phrase-type. The squares represent values between 0 and 1 by area.

I labeled each tree and its representation by the phrase type in the grammar, and sorted them by type. The RAAM has clearly developed a representation with similarity between members of the same type. For example, the third feature seems to be clearly distinguishing sentences from non-sentences, the fifth feature seems to be involved in separating adjective phrases from others, while the tenth feature appears to distinguish prepositional and noun phrases from others.[4]

At the same time, the representation must be keeping enough information about the subtrees in order to accurately reconstruct them.

The encoder and decoder networks thus form a recursive well-formedness test as follows: Take two trees, encode them into a new, higher-level, tree, and decode that back into two subtrees. If the subtrees are reconstructed within tolerance (i.e. τ for terminal binary vectors, and ν for non-terminal analog vectors) than that tree can be said to be well-formed.

The implications for conventional parsing is that instead of searching a grammar for a rule whose right-hand sides matches the two phrase-markers, we can accomplish this search with a simple feed-forward computation and parallel element-by- element comparison. This could lead to a somewhat nifty speed improvement.

The implications for connectionist parsing is that the output of an unsupervised RAAM could be the teacher for a supervised sequential learning technique such as sequential cascaded networks (Pollack, 1987b).

Furthermore, the well-formedness test can be used in a generator as follows. Start with a pool of trees composed of the terminals. Take every pair of trees from the pool, apply the well-formedness test, and if it passes, add the new higher-level tree to the pool.

Running this generator over the network formed from the above experiment yielded the following two interesting parse trees, among many silly ones:

(((D N)(P (D N)))(P (D N)))
((D (A N))(V (D N))

The first seems to be a recursive application of the NP -> NP PP rule, while the second is a single instance of a known NP newly appearing in a VP.

[4] In fact, by these metrics, the test case ((D N)(P (D N))) should really be classified as a sentence; since it was not used in any other construction, there was no reason for the RAAM to believe otherwise.

In my experiments thus far, RAAMS have only shown evidence of very limited generativity in this sense. The issue will be discussed further below.

DISCUSSION

The Tricks That Make RAAMS Work

Extensional Programming (Cottrell, et. al., 1987). Given that, with suitable control logic, the encoder produces a fixed-length analog representation for a sequence or tree which the decoder can decompose and reconstruct into a good facsimile of the original sequence or tree, then the fixed-length vector **must** be representing that sequence or tree.

Embedding the developing representations into the training environment. This is really new, though McClelland & St. John (1986), apparently tried something similar.

Recursively Reduced Descriptions. Recursive Autoassociative Memory appears to implement one of Hinton's (unpublished) idea's that has been floating around for a couple of years. With implementation, however, comes both validation and better understanding.

Constrained Co-evolution of Multiple Mechanisms. The encoder and decoder of a RAAM are really separate mechanisms which are evolved simultaneously and cooperate to develop a shared representation. There may be a useful foundational principle at work

Performance Induced from Tolerance Chaining. Neither stack nor tree mechanisms are really trained to be what they become. I am making an induction, backed up by lots of experimentation, that if an original encoded representation is compressed even further by the encoder, but can be reconstructed by the decoder to within a close tolerance of the original, then this reconstructed representation can be decoded almost as well as the original. It stands to reason that the error tolerance must get smaller as the desired representations get deeper, but this has not yet been quantified.

Compositionality is Settled

Fodor & Pylyshyn (1988) attacked the non-compositional nature of connectionist representations, while McDermott (1986) challenged the community to be able to represent and reason about the meaning of a complex sentence like: "She is more at home with her fellow students than with me, her advisor."

Compositionality and complexity are no longer crippling issues for connectionism. The representations that RAAM's develop may not be obvious, except in the simplest and most highly constrained cases, but they are compositional in the strictest sense.

Systematicity is not Settled

Unfortunately, Fodor & Pylyshyn's generative capacity issue, which they called "systematicity", still stands for the time being. There are essentially three possibilities, which will be explored in future work. Given that I have shown some limited generative capacity in the form of a small number of new useful representations, I believe that either the second, third, or both will be the case.

1 RAAM will not be able to develop systematicity;

2 Better training environments and better activation functions are needed; or

3 Critical mass, in terms of network size or training environment size, is needed.

Default Topology and Training is not Essential

There is no reason why only single level networks for the encoder and decoder should be used. Hidden units in the decoder as well as real thresholds will probably be necessary for a RAAM to develop a real analog stack representation, which may affect generative capacity[5].

On the other hand, this simple form of autoassociation may be directly solvable by linear methods: Cottrell, et. al. (1987) reported that their autoassociator essentially performs decomposition into principal components, while Bourlard & Kamp (1988) reported that autoassociative networks have direct and optimal algebraic solutions. Whether such methods can be implemented under the constraints of extreme locality, and whether they might extend to recursive autoassociation are still, however, open questions.

CONCLUSIONS

Implications: The Ultimate Capacity Of RAAMS

I do not have any closed analytical forms for the capacity of RAAMs. Somebody else will have to do that work. Given that is is not really an file-cabinet or content-addressable memory, but a memory for a gestalt of rules for recursive pattern compression and reconstruction, results such as Willshaw's (1981) and Hopfield's (1982) do not directly apply. Binary patterns are not being stored, so one cannot simply count how many.

I have considered the capacity of such a memory in the limit, however, where the actual functions and analog representations are not bounded by single linear transformations and sigmoids or by 32-bit floating point resolution.

[5] The ultimate irony here would be if the competence/performance distinction reduced to a problem in roundoff error.

Saund (1987) has investigated autoassociation as a method of *dimensionality reduction*, and asserted that, in order to work, the data must be constrained to form a small dimensional parametric surface in a larger dimensional space.

Consider just a 2-1-2 autoassociator. It is really a reconstructible mapping from data points on the unit square to unique points on the unit line. In order to work, the environment should define a parametric 1-dimensional curve in 2-space, perhaps a set of connected splines. As more and more data points need to be encoded, this parametric curve must get "curvier" to cover them. In the limit, it is no longer a 1-dimensional curve, but a space-filling curve with a fractional dimension. (At this point, however, the similarity between nearby points is lost.)

It is not yet the time to discuss the implications of *fractal representations* for AI: Just consider that the decomposition of meaning cannot really stop at a semantic primitive such as "MTRANS".

Applications: Inference By Association

If the reader almost believes that (1) complex AI-style representations can now be encoded into fixed-length analog vectors, which are compositional, similarity-based, and recursive in nature, and (2) that functions can be devised, say, using back-propagation, which perform arbitrary associations (that can generalize) between such fixed-width vectors, than it is not too hard a leap to believe that it may be possible, in the near future, to construct simple, constant-time, deterministic (but fallible) mechanisms for apparently rule-based processes such as parsing, logical deduction, plausible inference, and, perhaps, even, syntactic transformations.

I will be exploring several of these applications, with the slogan of "Associative Inference" over the next year.

ACKNOWLEDGEMENTS & REFERENCES

Tony Plate implemented back-propagation and the necessary modifications in C in exchange for a nice lunch; another is due for solving the traveling scientist problem (filling in the blanks of many references).

Ackley, D. H., Hinton, G. E. & Sejnowski, T. J. (1985). A learning algorithm for Boltzmann Machines. *Cognitive Science*, 9, 147-169.

Bourlard, H. & Kamp, Y. (1988) Auto-Association by Multilayer Perceptions and Singular Value Decomposition Technical Report M217. Brussels: Phillips Research Laboratory.

Cottrell, G. W. (1985). Connectionist Parsing. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*. Irvine, CA.

Cottrell, G., Munro, P., and Zipser, D. (1987). Learning Internal Representations from Gray-Scales Images: An Example of Extensional Programming. *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*. Seattle, WA. 461-473.

Fanty, M. (1985). Context-free parsing in Connectionist Networks. TR174, Rochester, N.Y.: University of Rochester, Computer Science Department.

Fodor J. & Pylyshyn, Z. (1988) Connectionism and Cognitive Architecture: A critical Analysis. *Cognition*, 28, 3-71.

Hanson, S. and Kegl, J. (1987) PARSNIP: A connectionist network that learns natural language grammar from exposure to natural language sentences. *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Seattle, WA., 106-119.

Hinton G. (unpublished). Representing Part-Whole Hierarchies in Connectionist Networks.

Hopfield, J. J. (1982). Neural Networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79, 2554-2558.

McClelland J. & St. John, M. (1986.) Reconstructive Memory for Sentences: A PDP Approach. *Proceedings of the Ohio University Inference Conference.*

McClelland, J. and Kawamoto A. (1986) Mechanisms of Sentence Processing: Assigning Roles to Constituents. In J. L. McClelland, D. E. Rumelhart & the PDP research group, (Eds.), *Parallel Distributed Processing: Experiments in the Microstructure of Cognition*, Vol. 2. Cambridge: MIT Press

McDermott, D. (1986) What AI needs from connectionism. Appendix to McClelland, J.L., Feldman, J.A., Bower, G. & McDermott, D. Connectionist models and cognitive science: goals, directions and implications. A report on an NSF workshop.

Mozer, M. (1984) Inductive Information Retrieval Using Parallel Distributed Computation. Technical Report. La Jolla: Institute for Cognitive Science, UCSD.

Pollack, J. B. (1987a) *Connectionism and natural language processing*. Cambridge: MIT Press. (In preparation; Available as MCCS-87-100, Computing Research Laboratory, NMSU, Las Cruces)

Pollack, J. B. (1987b) Cascaded back-propagation on dynamic connectionist networks. *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Seattle, WA.

Rosenfeld R. & Touretzky, D. (1987). Four capacity models for coarse-coded symbol memories. CMU-CS-87-182. Pittsburgh: Computer Science Dept., Carnegie-Mellon University.

Rumelhart, D. E., Hinton, G. & Williams, R. (1986). Learning Internal Representations through Error Propagation. In D. E. Rumelhart, J. L. McClelland & the PDP research Group, (Eds.), *Parallel Distributed Processing: Experiments in the Microstructure of Cognition,* Vol. 1. Cambridge: MIT Press.

Rumelhart, D. E. & McClelland, J. L. (1986). On Learning the Past Tenses of English Verbs. In J. L. McClelland, D. E. Rumelhart & the PDP research group, (Eds.), *Parallel Distributed Processing: Experiments in the Microstructure of Cognition*, Vol. 2. Cambridge: MIT Press.

Saund, E. (1987) Dimensionality Reduction and Constraint in Later Vision.*Proceedings of the Ninth Annual Conference of the Cognitive Science Society,*, 908-915, Seattle, WA.

Selman, B. (1985). Rule-Based Processing in a Connectionist System for Natural Language Understanding. CSRI-168, Toronto, Canada: University of Toronto, Computer Systems Research Institute.

Touretzky, D. S. & Hinton, G. E. (1985). Symbols among the neurons: details of a connectionist inference architecture. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence.* Los Angeles, CA.

Touretzky, D. S. (1986a). BoltzCONS: Reconciling connectionism with the recursive nature of stacks and trees. In *Proceedings of the 8th Annual Conference of the Cognitive Science Society.* Amherst, MA, 522-530.

Touretzky, D. S. (1986b). Representing and transforming recursive objects in a neural network, or ''trees do grow on Boltzmann machines''. In roceedings of the 1986 Institute of Electrical and Electronics Engineers International Conference on Systems, Man, and Cybernetics. Atlanta, GA.

Waltz, D. L. & Pollack, J. B. (1985). Massively Parallel Parsing: A strongly interactive model of Natural Language Interpretation. *Cognitive Science*, 9, 51-74.

Willshaw, D. (1981). Holography, Associative Memory, and Inductive Generalization. In Hinton, G. and J. Anderson (Eds.) *Parallel Models of Associative Memory.* Hillsdale: Lawrence Erlbaum Associates.