Recursive Distributed Representations

Jordan B. Pollack

Laboratory for AI Research &
Computer & Information Science Department
The Ohio State University
2036 Neil Avenue
Columbus, OH 43210
(614) 292-4890
pollack@cis.ohio-state.edu

ABSTRACT

A long-standing difficulty for connectionist modeling has been how to represent variable-sized recursive data structures, such as trees and lists, in fixed-width patterns. This paper presents a connectionist architecture which automatically develops compact distributed representations for such compositional structures, as well as efficient accessing mechanisms for them. Patterns which stand for the internal nodes of fixed-valence trees are devised through the recursive use of back-propagation on three-layer auto-associative encoder networks. The resulting representations are novel, in that they combine apparently immiscible aspects of features, pointers, and symbol structures. They form a bridge between the data structures necessary for high-level cognitive tasks and the associative, pattern recognition machinery provided by neural networks.

1. Introduction

One of the major stumbling blocks in the application of Connectionism to higher-level cognitive tasks, such as Natural Language Processing, has been the inadequacy of its representations. Both local and distributed representations have, thus far, been unsuitable for capturing the dynamically-allocated variable-sized symbolic data-structures traditionally used in AI. The limitation shows in the fact that pure connectionism has generated somewhat unsatisfying systems in this domain; for example, parsers for fixed length sentences [1-4], without embedded structures [5].[1]

Indeed, some of the recent attacks on connectionism have been aimed precisely at the question of representational adequacy. According to Minsky & Papert [10], for example, work on neural network and other learning machines was stopped by the need for AI to focus on knowledge representation in the 1970's, because of the principle that "no machine can learn to recognize X unless it possesses, at least potentially, some scheme for *representing* X (p. xiii)." Fodor and Pylyshyn's [11] arguments against connectionism are based on their belief that connectionist machines do not even have the *potential* for representing X, where X is combinatorial (syntactic) constituent structure, and hence cannot exhibit (semantic) "systematicity" of thought processes.

Agreeing thoroughly that compositional symbolic structures are important, in this paper I show a connectionist architecture which can discover compact distributed representations for them. *Recursive Auto-Associative Memory* (RAAM) uses back-propagation [12] on a non-stationary environment to devise patterns which stand for all of the internal nodes of fixed-valence trees. Further, the representations discovered are not merely connectionist implementations of classic concatenative data structures, but are in fact *new*, interesting, and potentially very useful.

The rest of this paper is organized as follows. After a background on connectionist representational schemes, the RAAM architecture is described, and several experiments presented. Finally, there is a discussion of the generative capacity of the architecture, and an analysis of the new representations and their potential applications.

---

[1] Hybrid (connectionist-symbolic) models [6-9] have the potential for more powerful representations, but do not insist on the neural plausibility constraints which create the limitations in the first place.

1.1. Background: Connectionist Representations

Normal computer programs have long used sequential data structures, such as arrays and lists as primitives. Because of the built-in notion of "address", moreover, the contents of sequences can be the addresses of other sequences; hence it is also quite simple for computer programs to represent and manipulate tree and graph structures as well. Representing lists and trees is not a trivial problem for connectionist networks, however, which do not use adjacent or randomly addressed memory cells, or permit the real-time dynamic creation of new units.

Some of the earliest work in modern connectionism made an inappropriate analogy between semantic networks and neural networks. The links in the former represented logical relations between concepts. The links in the latter represented weighted paths along which "activation energy" flowed. Needless to say, these first connectionist networks, in which each concept was mapped onto a single neuron-like unit, did not have the representational capacity of their logically powerful cousins.

Furthermore, local representational schemes do not efficiently represent sequential information. The standard approach involves converting time into space by duplicating sub-networks into a fixed set of buffers for sequential input. Both early connectionist work, such as McClelland & Rumelhart's word recognition model [13], as well as more modern efforts [4, 14] use this approach, which is not able to represent or process sequences longer than a predetermined bound. One way to overcome this length limitation is by "sliding" the input across the buffer [15, 16]. While such systems are capable of processing sequences longer than the predetermined bound, they are not really representing them.

Distributed Representations have been the focus of much research (including the work reported herein) since the circulation of Hinton's 1984 report [17] discussing the properties of representations in which "each entity is represented by a pattern of activity distributed over many computing elements, and each computed element is involved in representing many different entities."

The most obvious and natural distributed representation is a feature (or microfeature) system, traditionally used in linguistics. A good example of a connectionist model using such a representation is Kawamoto's work on lexical access [18]. However, since the entire feature system is needed to represent a single concept, attempts at representing structures involving those concepts cannot be managed in the same system. For example, if all the features are needed to represent a NURSE, and all the features are needed to represent an ELEPHANT, then the attempt to represent a NURSE RIDING ELEPHANT may come out either as a WHITE ELEPHANT or a rather LARGE NURSE WITH FOUR LEGS.

To solve the problem of feature superposition, one might use full-size constituent buffers, such as Agent, Action, and Object [5]. In each buffer would reside a feature pattern filling these roles such as NURSE, RIDING, and ELEPHANT. Unfortunately, because of the dichotomy between the representation of a structure (by concatenation) and the representation of an element of the structure (by features), this type of system cannot represent embedded structures such as "John saw the nurse riding an elephant." A solution to the feature-buffer dichotomy problem was anticipated and sketched out by Hinton [19], and involved having a "reduced description" for NURSE RIDING ELEPHANT which would fit into the constituent buffers along with patterns for JOHN and SAW.

However, it was not immediately obvious how to develop such reduced descriptions. Instead, avant-garde connectionist representations were based on coarse-coding [17], which allows multiple semi-independent representational elements to be simultaneously present, by superposition, in a feature vector. Once multiple elements can be present, conventional groupings of the elements can be interpreted as larger structures.

For example, Touretzky has developed a coarse-coded memory system and used it in a production system [20], a primitive lisp data-structuring system called BoltzCONS [21], and a combination of the two for simple tree manipulations [22]. In his representation, the 15,625 triples of 25 symbols (A-Y) are elements to be represented, and using patterns over 2000 bits, small sets of such triples could be reliably represented. Interpreting the set of triples as pseudo-CONS cells, a limited representation of sequences and trees could be achieved.

Similarly, in their past-tense model, Rumelhart and McClelland [23] developed an *implicitly sequential representation*, where a set of well-formed overlapping triples could be interpreted as a sequence. It is instructive to view the basic idea of their representational scheme as the encoding of a sequence of tokens, $(i_1, \cdots, i_n)$ by an unordered *set* of overlapping subsequences (each of breadth k) of tokens:

$$\{(i_1, \cdots, i_k), (i_2, \cdots, i_{k+1}), \cdots, (i_{n-k+1}, \cdots, i_n)\}$$

Thus, if a coarse-coded memory can simultaneously represent a set of such subsequences, then it can also represent a longer sequence.

The limits of this type of representation are that the cost of the representation goes up exponentially with its breadth, and, for any particular breadth, there may be sequences with too much internal duplication. Sets do not count multiple occurrences of their elements. So a system, for example, which represented the spellings of words as sets of letter-*pairs* would not be able to represent the word *yoyo*, and even if the breadth were increased to three, the system would still not be able to represent words with duplicate triples such as *banana*.[2]

Although both Touretzky's and Rumelhart & McClelland's coarse-coded representations were fairly successful for their circumscribed tasks, there remain some problems:

(1) A large amount of human effort was involved in the design, compression and tuning of these representations, and it is often not clear how to translate that effort across domains.

(2) Coarse-coding requires expensive and complex access mechanisms, such as pullout networks [25] or clause-spaces [20].

(3) Coarse-coded symbol memories can only simultaneously instantiate a small number of representational elements (like triples of 25 tokens) before spurious elements are introduced[3]. Furthermore, they assume that all possible tokens need to be combined.

(4) They utilize binary codes over a large set of units (hundreds or thousands).

(5) Their mode of aggregating larger structures out of basic elements is superpositional, the cause of problems (2) and (3).

In contrast, the distributed representations devised by the RAAM architecture demonstrate better properties:

(1) Encodings are developed mechanically by an adaptive network.

---

[2] To point out this "Banana Problem" with Rumelhart & McClelland's actual representation, which was phonological rather than orthographic, Pinker and Prince [24] discovered words with enough internal duplication in the Oykangand language.

[3] Rosenfeld and Touretzky [26] provide a nice analysis of coarse-coded symbol memories.

(2) The access mechanisms are simple and deterministic.

(3) A potentially very large number of primitive elements can *selectively* combine into constituent structures. Not all triples of symbols can, or need, be represented.

(4) The representations utilize real-values over few units (tens).

(5) The aggregation mode is compositional.

2. Recursive Auto-Associative Memory

The problem under attack, then, is the representation of variable-sized symbolic sequences or trees in a numeric fixed-width form, suitable for use with association, categorization, pattern-recognition, and other neural-style processing mechanisms.

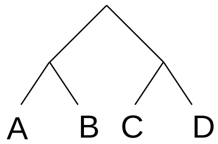

Figure 1. *Example of a binary tree.*

Consider two hypothetical mechanisms which could translate, in both directions, between symbolic trees and numeric vectors. The *Compressor* should encode small sets of fixed-width patterns into single patterns of the same size. It could be recursively applied, from the bottom up, to a fixed-valence tree with labeled terminals (leaves), resulting in a fixed-width pattern representing the entire structure. For the binary tree *((A B)(C D))*, shown in figure 1, where each of the terminals is a fixed-width pattern, this would take three steps. First A and B would be compressed into a pattern, R_1. Then C and D would be compressed into a pattern, R_2. Finally, R_1 and R_2 would be compressed into R_3.

The *Reconstructor* should decode these fixed-width patterns into facsimiles of their parts, and determine when the parts should be further decoded. It could be recursively applied, from the top down, resulting in a reconstruction of the original tree. Thus, for this example, R_3 would be decoded into R_1' and R_2'. R_1' would be decoded into A' and B', and R_2' into C' and D'.

These mechanisms are hypothetical, because it is not clear either how to physically build or computationally simulate such devices, or what the R_i patterns look like. In answer to the first question, I just assume that the mechanisms could be built out of the

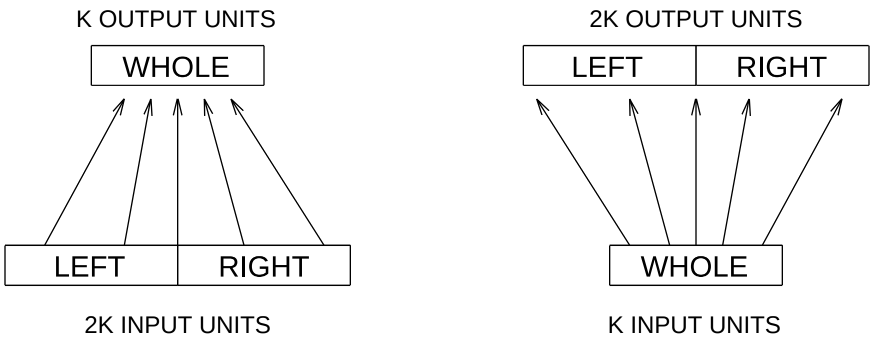

Figure 2. *Proposed feedforward networks for the Compressor and Reconstructor working with binary trees.*

standard modern connectionist substrate of layered fully-connected feed-forward networks of semi-linear units.[4] For binary trees with k-bit patterns as the leaves, the compressor could be a single-layer network with $2k$ inputs and k outputs. The reconstructor could be a single-layer network with k inputs and $2k$ outputs. Schematics for these are shown in Figure 2.

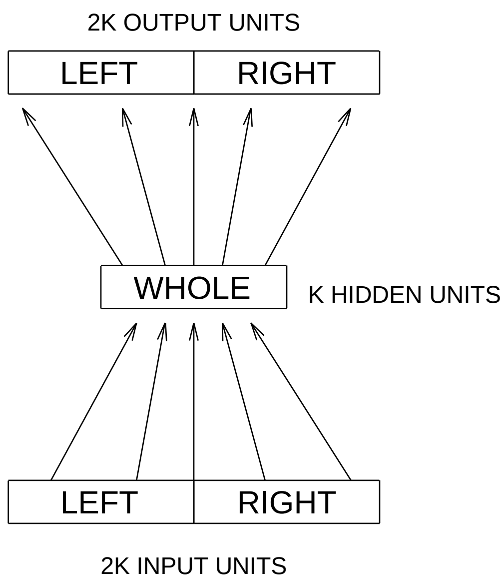

Figure 3. *Single network composed of both compressor and reconstructor.*

In answer to the second, regarding what the patterns look like, we develop the strategy of letting a connectionist network devise its own representations. Consider

[4] I also assume that the reader is, by now, familiar with this standard, as well as with the back-propagation technique for adjusting weights [12], and will not attempt a re-presentation of the mathematics. The work herein does not crucially depend on the default assumptions of semi-linearity and full-connectedness. By relying on these standard defaults, however, I hope to keep the focus on issue of representation.

simultaneously training these two mechanisms as a single $2k-k-2k$ network, as shown in Figure 3.

This looks suspiciously like a network for the Encoder Problem [27]. Back-propagation has been quite successful at this problem,[5] when used in a self-supervised auto-associative mode on a three layer network. The network is trained to reproduce a set of input patterns; i.e., the input patterns are also used as desired (or target) patterns. In learning to do so, the network develops a compressed code on the hidden units for each of the input patterns. For example, training an 8-3-8 network to reproduce the eight 1-bit-in-8 patterns usually results in a 3-bit binary code on the hidden units.

In order to find codes for trees, however, this auto-associative architecture must be used recursively (hence its name). Extending the simple example from above, if A, B, C, and D were k-bit patterns, the network could be trained to reproduce (A B), (C D), *and* ((A B)(C D)) as follows:

input pattern		*hidden pattern*		*output pattern*
$(A \quad B)$	$\rightarrow$	$R_1(t)$	$\rightarrow$	$(A'(t) \quad B'(t))$
$(C \quad D)$	$\rightarrow$	$R_2(t)$	$\rightarrow$	$(C'(t) \quad D'(t))$
$(R_1(t) \quad R_2(t))$	$\rightarrow$	$R_3(t)$	$\rightarrow$	$(R_1(t)' \quad R_2(t)')$

where t represents the time, or epoch, of training. Assuming that back-propagation converges in the limit, the sum of the squares of the differences between the desired and actual outputs would go to 0, and:

$$\begin{aligned} A' &= A \\ B' &= B \\ C' &= C \\ D' &= D \\ R_1' &= R_1 \\ R_2' &= R_2 \end{aligned}$$

Therefore, R_3, would, in fact, be a representation for the tree ((A B)(C D)), by virtue of the fact that the compressor would be a deterministic algorithm which transforms the tree to its representation, and the reconstructor a deterministic algorithm which transforms the representation back to the tree. Along the way, representations will also be devised for all subtrees, in this case, (A B) and (C D). Note that, as will be

[5] Rumelhart et al. [12] demonstrated only a 8-3-8 network, but other successful uses include a 64-16-64 network [28] and a 270-45-270 network [4]. The three numbers correspond to the number of units in the input, hidden, and output layers of a network.

demonstrated later, this strategy works on a collection of trees just as it does on a single tree.

There are a few details which form a bridge between theory and practice.

(1) The (initially random) values of the hidden units, $R_i(t)$, are used as part of the training environment. Therefore, as the weights in the network evolve, so do some of the patterns that comprise the training environment. This form of non-stationary, or ‘‘Moving Target,’’ learning has also been explored by others [29, 30]. The stability and convergence of the network are sensitive to the learning parameters. Following the explication of Rumelhart et al. [12, p. 330], there are two such parameters: the learning rate η, which controls the the gradient descent step size, and the momentum α, which integrates the effects of previous steps. These parameters must be set low enough that the change in the hidden representations does not invalidate the decreasing error granted by the change in weights, and high enough that some change actually takes place. In the experiments described later in this paper, η was usually set to 0.1 (less for the larger experiments), and α to 0.3. As the learning curve flattens out, α is slowly increased up to 0.9, following [31].

(2) The induction relied upon is outside the mechanical framework of learning. This induction, of global success arising from only local improvements, is similar to the Bucket Brigade principle used in classifier systems [32]. Since the training strategy never reconstructs the terminals from R'_1 or R'_2, only the fact that they are equal, in the limit, to R_1 and R_2 allows this strategy to work.

But back-propagation cannot really run forever, and therefore, at least with use of the standard sigmoidal activation function, it is impossible to achieve the perfect encoding described above. So some practical way to decide when to stop training becomes necessary. When back-propagation is used to produce binary outputs, there is a tolerance, τ, conventionally set to 0.2, such that training can stop when every output value for every training pattern is within τ of the desired bit. For non-terminal patterns which may not be binary, however, 20% is far too permissive a tolerance. In order to successfully reconstruct A and B (to a tolerance of τ) from R'_1, for example, R'_1 must be *very* similar to R_1. Thus, a second tolerance, ν, is used for the real-valued non-terminals, which, for the experiments below, has been set at 0.05.

(3) The name for this architecture, Recursive Auto-Associative Memory (RAAM), accurately reflects that the codes developed by an auto-associative memory are being further compressed. It does not reflect that there are actually two separate mechanisms which happen to be simultaneously trained. These mechanisms also require some support in the form of control and memory, but nothing beyond the ability of simple neural networks using thresholds.

In order to encode a tree from the bottom up, the compressor needs a stack on which to store temporary results (such as R_1). In order to decode a tree from the top down, the reconstructor also needs an external stack on which to store intermediate patterns. Furthermore, it needs some mechanism to perform terminal testing. In the experiments presented below, it is assumed that this terminal test is merely a threshhold test for "binary-ness", which checks that all the values of a pattern are above $1-\tau$ or below τ. Alternatively, one could train a simple classifier, or use conventional computer programs which test for membership in a set, or perform error detection and correction.

2.1. Sequential RAAM

Since sequences, such as (X Y Z), can be represented as left-branching binary trees, i.e., ((NIL X) Y) Z), an alternative version of the RAAM architecture works for developing representations and *Last-In-First-Out* access mechanisms for sequences.

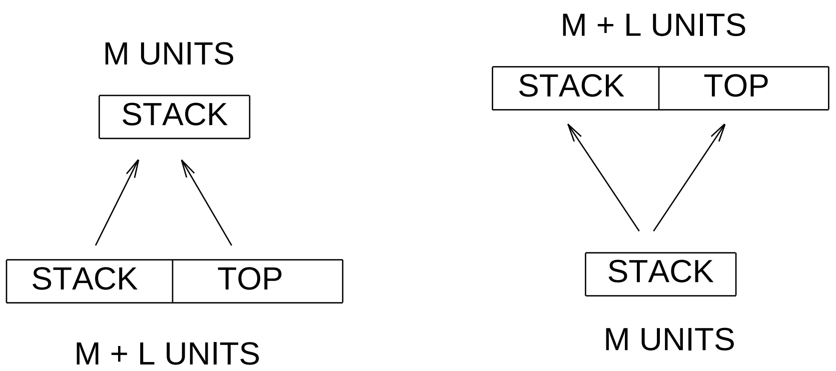

Figure 4. *Inverse sequencing mechanisms in single-layered networks. The compressor combines an m-dimensional representation for a sequence (STACK) with a new element (TOP), returning a new m-dimensional vector;*

the reconstructor decodes it back into its components.

This architecture is in fact simpler than the mechanism for trees. Compressed representations only have to be recirculated to one side, so they do have to be stored externally. There is less constraint on the size of the representations as well, and a higher dimension, M, can be assumed for the compressed patterns, than for the terminal symbols, L.

Figure 4 shows the single-layer compressor and reconstructor networks for a sequential RAAM, which, when viewed as a single network has $M+L$ input and output units, and M hidden units. An M-vector of numbers, ε, is assumed to stand for NIL, the empty sequence. In the experiments below, vectors of all 0.5's are chosen, which are very unlikely ever to be generated as an intermediate state. Following the earlier logic, when this network is trained with the patterns:

input pattern		*hidden pattern*		*output pattern*
$(\varepsilon \quad X)$	$\rightarrow$	$R_x(t)$	$\rightarrow$	$(\varepsilon'(t) \quad X'(t))$
$(R_x(t) \quad Y)$	$\rightarrow$	$R_{xy}(t)$	$\rightarrow$	$(R'_x(t) \quad Y'(t))$
$(R_{xy}(t) \quad Z)$	$\rightarrow$	$R_{xyz}(t)$	$\rightarrow$	$(R_{xy}(t)' \quad Z'(t))$

it is expected that, after back-propagation converges, R_{xyz} will be a representation for the sequence (X Y Z). Along the way, representations will also be developed for all prefixes to the sequence, in this case, (X) and (X Y).

3. Experiments with Recursive Auto-Associative Memories

3.1. Proof of Concept

To demonstrate that RAAM actually works under practical assumptions, and that it can discover compositional representations and simple access mechanisms, a small sequential RAAM is presented first.

The training set consisted of the eight possible sequences of three bits. Using a 4-3-4 network and an empty pattern of (.5 .5 .5), the representations shown in Figure 5 were developed. (The representations for all the prefixes are shown as well). The network has clearly developed into a tri-state shift-register, where the first feature corresponds to the inverse of the last bit in, the second to the inverse of the next-to-last bit, and the third to the first bit encoded.

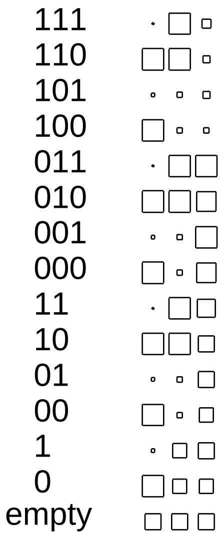

Figure 5. *Representations developed by a 4-3-4 RAAM for the complete set of bit patterns up to length 3. Each square represents a number between 0 and 1.*

A shift-register, which simply concatenates bits, is a classical means for serially constructing and accessing an obviously compositional representation. But like any finite piece of hardware built to hold a certain number of bits, it degrades rather rapidly when over-filled. The more interesting area to explore involves pattern spaces which have underlying regularities, but do not depend on representing all possible combinations of sub-patterns. It is under these conditions that an adaptive connectionist mechanism would be expected to display more desirable properties, such as content-sensitivity and graceful degradation.

3.2. Letter Sequences

Our second experiment involves learning to represent sequences of letters. Rather than trying to represent all possible sequences of letters, which would certainly give rise to another shift register, a limited subset of English words was chosen. Using an electronic spelling dictionary, those words containing only the 5 letters "B", "R", "A", "I", and "N" were selected, and then all prefixes (like "an" and "bar") were removed, resulting in the list below. Note that, in training, a representation is developed for every prefix:

AIR	ANA	ANI	BABAR	BANANA
BARBARIAN	BARN	BIBB	BIN	BRAIN
BRAN	BRIAR	INN	NAB	NIB
RABBI	RAIN	RAN	RIB	

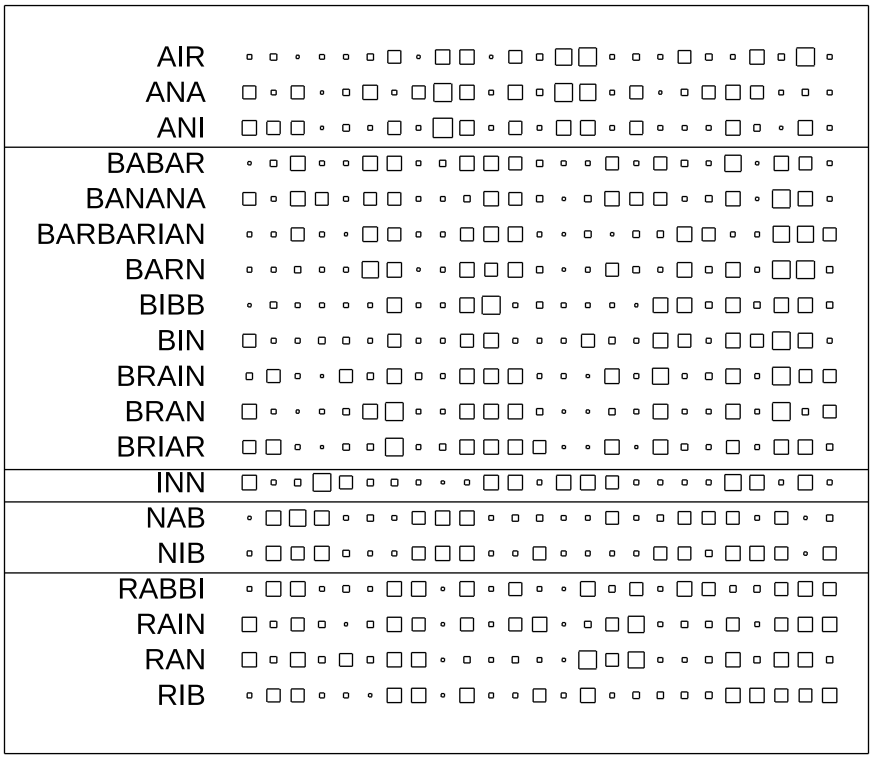

Figure 6. *Representations developed by a 30-25-30 RAAM on letter sequences.*

Each terminal was coded as a 1-in-5 bit pattern, the empty vector, again, was all 0.5's, and a 30-25-30 RAAM was used to encode these words. Note that both BANANA and BARBARIAN would be troublesome for an implicit sequential representation of breadth three. Figure 6 shows the representations for these letter sequences, and the cluster diagram in Figure 7 shows that, unlike a decaying sum representation in which information about older elements gets lost [33], this sequential representation is devoting the most resources to keeping older elements alive. And even though there are enough resources to build a 5-letter shift register, the network cannot take this easy solution path because of its need to represent the 6- and 9-letter words.

```
 /------> INN
-|       ______/-----> AIR
 |      /      \_____/----> ANA
 \------|            \----> ANI
        |       ______/-----> NAB
        |      /      \-----> NIB
        \------|       _____/-----> RAIN
               |      /     \_____/-----> RAN
               \------|           \_____/-----> RIB
                      |                 \-----> RABBI
                      |      _____/-----> BIN
                      \-----/     \-----> BIBB
                            |     /-----> BRIAR
                            \-----|      _____/----> BRAN
                                  \-----/     \----> BRAIN
                                        \_____/-----> BABAR
                                              \_____/-----> BANANA
                                                    \_____/----> BARN
                                                          \----> BARBARIAN
```

Figure 7. *Hierarchal clustering of the letter sequence representations.*

3.2.1. Learning Well-formed Syntactic Trees

The tree ((D (A N))(V (P (D N)))) might be a syntactic parse-tree for the sentence "The little boy ran up the street", given that the terminals D, A, N, V, and P stand respectively for *determiner*, *adjective*, *noun*, *verb,* and *preposition*. Consider a simple context-free grammar, where every rule expansion has exactly two constituents:

```
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, it is easy to derive the bracketed binary trees which will make up a training set. With one such set of strings, a chart parser yielded 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))))
```

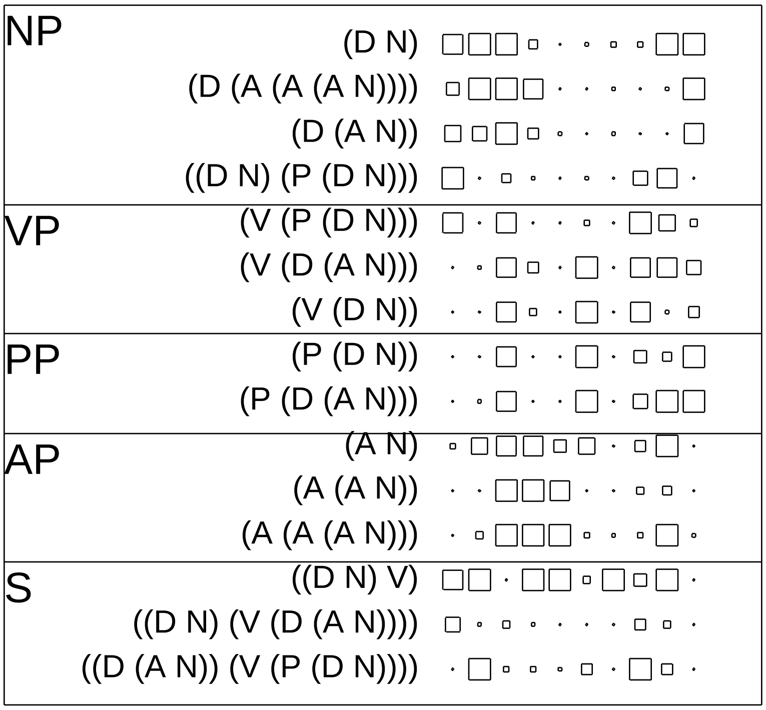

Figure 8. *Representations of all the binary trees in the training set, devised by a 20-10-20 RAAM, manually clustered by phrase-type.*

Each terminal (D A N V & P) was then represented as a 1-bit-in-5 code padded with 5 zeros. A 20-10-20 RAAM devised the representations shown in Figure 8.

```
  _________/--------> (AN)
 /         \--------> (DN)
 |                   ________/-------> (P(DN))
 |                  /        \-------> (P(D(AN)))
-|         /--------|         ________/--------> (V(DN))
 |         |        |        /        \________/-------> (V(D(AN)))
 |         |        \--------|                 \-------> (V(P(DN)))
 |         |                 |         ________/-------> (A(AN))
 \---------|                 \--------/        \-------> (A(A(AN)))
           |                          \________/------> (D(AN))
           |                                   \------> (D(A(A(AN))))
           |         _______/-------> ((DN)V)
           \--------/       \-------> ((DN)(P(DN)))
                    \_______/-------> ((DN)(V(D(AN))))
                            \-------> ((D(AN))(V(P(DN))))
```

Figure 9. *Hierarchal clustering of the syntactic patterns.*

Each tree and its representation have been labeled by the phrase type in the grammar, and then sorted 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 almost separates adjective phrases from others, while the tenth feature appears to distinguish prepositional and noun phrases from the rest.[6] Finally, a hierarchal cluster of these patterns in Figure 9 reveals that the similarity between patterns generally follows the phrase type breakup, and also reflects the depth of trees.

3.2.2. Learning to Represent Propositions.

Tree representations are common data structures, used for semantic as well as syntactic structures. This final experiment sets up some propositional representations which will be exploited later in the paper, and merely demonstrates that the architecture is capable of working on more than just binary trees.[7]

Table 1. *Collection of sentences for propositional experiment.*

1	Pat loved Mary
2	John loved Pat
3	John saw a man on the hill with a telescope
4	Mary ate spaghetti with chopsticks
5	Mary ate spaghetti with meat
6	Pat ate meat
7	Pat knew John loved Mary
8	Pat thought John knew Mary loved John
9	Pat hoped John thought Mary ate spaghetti
10	John hit the man with a long telescope
11	Pat hoped the man with a telescope saw her
12	Pat hit the man who thought Mary loved John
13	The short man who thought he saw John saw Pat

Starting with a somewhat random collection of sentences, a RAAM was used to devise compact representations for corresponding propositional forms. The sentences used for training are shown in Table 1. The terminals for this RAAM are bit patterns for the symbols which appear in these sentences minus the determiners and pronouns, plus two new symbols: *IS* is used as a subject-raiser in the representations for sentences 11 and 12, while *MOD* is used to specify adjectives in triples.

---

[6] By these metrics, of course, ((D N)(P (D N))) is being classified as an S rather than an NP. This is not surprising since, like an S, it is not being further combined.

[7] Of course, binary trees of symbols (along with a distinguished NIL element) are sufficient for arbitrary tree representations.

Table 2. *16-bit patterns for the terminal symbols*

WORD	THING 4 BITS	HUMAN 3 BITS	PREP 3 BITS	ADJ 2 BITS	VERB 4 BITS
HILL	1 0 0 0				
STREET	1 0 0 1				
TELESCOPE	1 0 1 0				
CHOPSTICKS	1 0 1 1				
MEAT	1 1 0 0				
SPAGHETTI	1 1 0 1				
MAN		1 0 0			
JOHN		1 0 1			
MARY		1 1 0			
PAT		1 1 1			
MOD			1 0 0		
WITH			1 0 1		
ON			1 1 0		
LONG				1 0	
SHORT				1 1	
IS					1 0 0 0
KNEW					1 0 0 1
HOPED					1 0 1 0
THOUGHT					1 0 1 1
LOVED					1 1 0 0
HIT					1 1 0 1
ATE					1 1 1 0
SAW					1 1 1 1

Table 3. *Ternary trees for propositional experiment.*

1	(LOVED PAT MARY)
2	(LOVED JOHN PAT)
3	((WITH SAW TELESCOPE) JOHN (ON MAN HILL))
4	((WITH ATE CHOPSTICKS) MARY SPAGHETTI)
5	(ATE MARY (WITH SPAGHETTI MEAT))
6	(ATE PAT MEAT)
7	(KNEW PAT (LOVED JOHN MARY))
8	(THOUGHT PAT (KNEW JOHN (LOVED MARY JOHN)))
9	(HOPED PAT (THOUGHT JOHN (ATE MARY SPAGHETTI)))
10a	((WITH HIT (MOD TELESCOPE LONG)) JOHN MAN)
10b	(HIT JOHN (WITH MAN (MOD TELESCOPE LONG)))
11	(HOPED PAT (SAW (WITH MAN TELESCOPE) PAT))
12	(HIT PAT (IS MAN (THOUGHT MAN (LOVED MARY JOHN))))
13	(SAW (IS (MOD MAN SHORT) (THOUGHT MAN (SAW MAN JOHN))) PAT)

A similarity-based 16-bit binary representation was devised for the terminals, by first dividing them into 5 classes, *THING*, *HUMAN*, *PREP*, *ADJ*, and *VERB*, and then using one bit for each class along with a counter as shown in Table 2. Empty spots are all zeros. Each sentence was manually translated into a ternary tree (except sentence 10

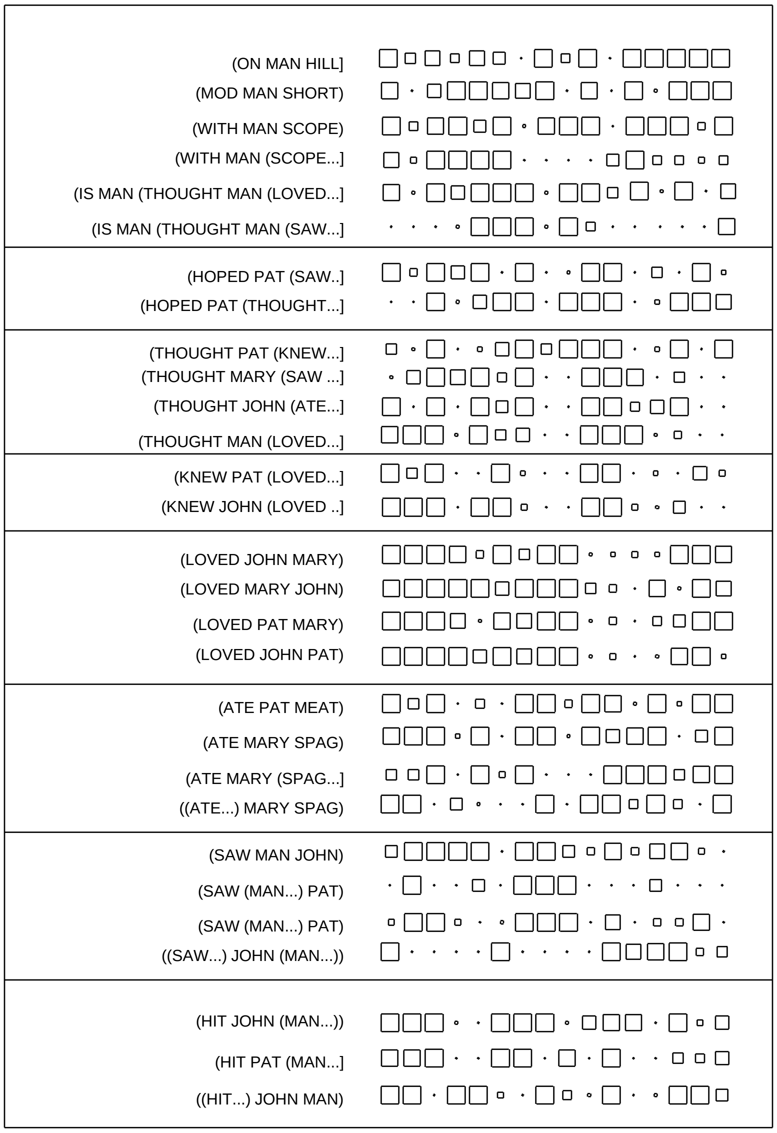

Figure 10. *Representations of the ternary semantic trees in the training set, devised by a 48-16-48 RAAM, manually clustered. The symbolic trees have been abbreviated to fit.*

which had two readings) as shown in Table 3. This representation is meant to capture the flavor of a recursive (ACTION AGENT OBJECT) case system. A 48-16-48 RAAM learned to construct representations and to recursively encode and decode these trees into their respective parts. These are again shown both pictorially (Figure 10) and clustered (Figure 11).

```
 /-----> (WITH HIT (MOD TELESCOPE LONG))
-|     /-----> (IS (MOD MAN SHORT) (THOUGHT MAN (SAW MAN JOHN)))
 \-----|     /-----> ((WITH SAW TELESCOPE) JOHN (ON MAN HILL))
       |     |     /-----> (ATE MARY (WITH SPAGHETTI MEAT))
       \-----|     |           ____/---> (HOPED PAT (THOUGHT JOHN (ATE MARY SPAGHETTI)))
             |     |          /    \---> (HOPED PAT (SAW (WITH MAN TELESCOPE) PAT))
             |     |     /----|         /----> (HIT JOHN (WITH MAN (MOD TELESCOPE LONG)))
             \-----|     |    |    /----|         ___/---> (KNEW PAT (LOVED JOHN MARY))
                   |     |    |    |    |    /---/   \---> (THOUGHT PAT (KNEW JOHN (LOVED MARY JOHN)))
                   |     |    \----|    \----|   \---> (KNEW JOHN (LOVED MARY JOHN))
                   |     |         |         \---> (HIT PAT (IS MAN (THOUGHT MAN (LOVED MARY JOHN))))
                   |     |         \____/---> (IS MAN (THOUGHT MAN (LOVED MARY JOHN)))
                   \-----|              \___/---> (THOUGHT JOHN (ATE MARY SPAGHETTI))
                         |                  \___/---> (THOUGHT MAN (LOVED MARY JOHN))
                         |                      \---> (THOUGHT MAN (SAW MAN JOHN))
                         |     ____/----> ((WITH ATE CHOPSTICKS) MARY SPAGHETTI)
                         |    /    \____/----> (ATE MARY SPAGHETTI)
                         |    |         \----> (ATE PAT MEAT)
                         |    |           _____/---> (WITH SAW TELESCOPE)
                         \----|    /-----/     \---> (WITH ATE CHOPSTICKS)
                              |    |     |     /-----> (WITH SPAGHETTI MEAT)
                              |    |     \-----|     /----> (WITH MAN (MOD TELESCOPE LONG))
                              |    |           \-----|     ____/----> (ON MAN HILL)
                              \----|                 \----/    \----> (WITH MAN TELESCOPE)
                                   |                      \____/----> (MOD TELESCOPE LONG)
                                   |                           \----> (MOD MAN SHORT)
                                   |     /-----> ((WITH HIT (MOD TELESCOPE LONG)) JOHN MAN)
                                   |     |                ___/--> (LOVED JOHN MARY)
                                   \-----|          /----/   \--> (LOVED PAT MARY)
                                         |     /----|    \---> (LOVED JOHN PAT)
                                         \-----|    \----> (LOVED MARY JOHN)
                                               |     _____/---> (SAW (WITH MAN TELESCOPE) PAT)
                                               \----/     \---> (SAW (IS (MOD MAN SHORT) (THOUGHT MAN (SAW MAN JOHN))) PAT)
                                                    \-----> (SAW MAN JOHN)
```

Figure 11. *Hierarchal clustering of the semantic patterns*

4. Discussion

4.1. Studies of Generalization

Perhaps the most important question about Recursive Auto-Associative Memories is whether or not they are capable of any productive forms of generalization. If it turned out that, as in the shift-register example, they were just *memorizing* the training set, finding a convenient mapping from given structures to unassigned vertices in a high-dimensional hypercube, then this work would ultimately be uninteresting. Luckily, this turns out not to be the case.

It is a straightforward matter to enumerate the set of sequences or trees that a RAAM is capable of representing, beyond the training set. Taken together, the encoder and decoder networks form a recursive well-formedness test as follows: Take two patterns for trees, encode them into a pattern for the new, higher-level, tree, and decode that back into the patterns for the two sub-trees. If the reconstructed subtrees are within tolerance, then that tree can be considered well-formed.[8]

Using this procedure for tree RAAMs, a program can start with the set of terminals as the pool of well-formed patterns, and then exhaustively (or randomly) combine all pairs, adding new well-formed patterns to the pool. For sequential RAAMs, the pool is begun with just the pattern for the empty sequence, and a program merely attempts to compose each terminal with each pattern in the pool, adding new prefixes to the pool as they are found..

Running this generator over the network formed from the syntactic tree experiment yielded 31 well-formed trees, which are shown in Table 4. Of these, the first 12 are not really grammatical, although 8 of these seem to be based on a rule which allows two NP's to combine. There are three new instances of NP's, four new VP's, and twelve new S's. Clearly some sort of generativity, beyond memorization, is going on here, though not yet in an infinite manner. At the least, new instances of the syntactic classes are being formed by recombination of parts.

The sequential RAAM for letter sequences is quite a bit more productive. It is able to represent about 300 new sequences of letters, of which approximately one-third are wordlike, including names not in the electronic spelling dictionary like BRIAN, RINA, and BARBARA. Mostly, however, the novel sequences reflect low-order letter-transition statistics, indicating, again, that some recollective process more powerful than rote (list) memorization but less powerful than arbitrary random-access sequential storage is taking place.

There is also a tendency, especially by the 48-16-48 RAAM, to decode novel trees back to existing members of the training set. For example, the pattern encoded for (THOUGHT JOHN (KNEW PAT (LOVED MARY JOHN))) is reconstructed to (THOUGHT PAT (KNEW JOHN (LOVED MARY JOHN))), one of the original trees.

This lack of productivity is probably attributable to the problem that the input patterns are *too* similar; i.e., the Hamming distance between JOHN and PAT is only one bit. But, while this RAAM was not as productive as hoped for, it was still quite systematic,

[8] Actually, this is a bit of a simplification, since the well-formedness test does not actually guarantee that the pattern for new tree can be fully decoded. If the tolerance is kept low enough, however, the full tree will be recoverable.

Table 4. *Additional trees that can be represented by the 20-10-20 RAAM*

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

according to Fodor & Pylyshyn's [11, p. 39] own definition:

> *What does it mean to say that thought is systematic? Well, just as you don't find people who can understand the sentence 'John loves the girl' but not the sentence 'the girl loves John,' so too you don't find people who can think the thought that John loves the girl but can't think the think the thought that the girl loves John.*

All 16 cases of (LOVED X Y), with X and Y chosen from the set {JOHN, MARY, PAT, MAN} were able to be reliably represented, even though only four of them were in the training set.

4.1.1. Improving Generalization Capacity

The productive capacity of these systems is not yet what it should be. There ought to be some way to acquire, at least theoretically, the ability to represent infinite numbers of similar structures in such recursive distributed representations.

Given that the simplest formulation (i.e., a 3-layer fully-connected semi-linear network) using rather arbitrary training sets has shown some limited capacity in the form of a small number of new useful representations composed out of existing constituents, it seems likely that (1) better training environments and (2) different mathematical assumptions will be needed.

First, the similarity and difference relationships between terminal patterns affects the productivity of a RAAM. In the case of the semantic triples, the fact that terminals in the same class, like JOHN and MARY, were assigned very similar patterns, lead both to their ability to be used systematically, and to the problem that single-bit errors in reconstruction were damaging. On the other hand, one would expect fully random patterns to not generalize very well either. This brings up the question of how to design compressible representations. It seems very likely that the same sort of representations devised by a RAAM for the non-terminal patterns would lead to the best possible compression and generalization properties if adopted for terminals.

Secondly, to achieve truly infinite representational capacity in fixed-width patterns, it will be necessary, at least theoretically, to consider the underlying mathematical basis for connectionist networks, freed from the default implementational assumptions of back-propagation, i.e., floating-point calculations of linear combinations and sigmoids. On the one hand, it must be considered whether or not to use real numbers at all since they are seem biologically and computationally problematic. An unbounded number of bits can be trivially compressed into a real number, leading to unbounded storage and communication costs. A simulated connectionist system using real numbers might be able to use these bits, (i.e. in very precise output values) without properly paying for them. By using only a binary code, a system must be able to to exploit the redundancy (i.e. sparseness or regularity) in the environment. On the other hand, it is certainly reasonable, however unbiological, to assume rational numbers for a competence theory. The question to answer is whether there is a similarity-preserving mapping from complex structured representations to high-dimensional spatial representations.

4.2. Analysis of the Representations

I do not yet have a prescription for engineering recursive distributed representations, but have a few insights into how they work. Top-down and bottom-up constraints work together to forge the representations. The bottom-up constraint is that each pattern is completely determined by its constituents and the knowledge eventually fixed in the network weights: *Trees with similar constituents must be similar.* The top-down constraint is that redundant information must be compressed out of similar structures (such as two NP's which both can combine with the same VP): *The possible siblings of a pattern must be similar.* Working against this drive towards similarity is the system-wide goal of minimizing error, which serves to "constrain apart" the patterns for different trees in the environment. The result of these pressures is that these representations consist of at least two types of features: *Categorical* features, such as those identified earlier as being able to separate classes, and *distinctive* features, which vary across, and discriminate between, the members of each class.

The categorical features developed by the syntactic tree experiment become clear in examination of the of a small classifier. The patterns for each tree in the training set were used as input to a 10-input 5-output network which was trained to discriminate the classes NP, VP, PP, AP, and S.

Table 5. *Weights of single-layer classifier network rounded to integers.*

	NP	VP	PP	AP	S	Strength
Bias	-2	-8	-3	-4	6	
1	8	0	-2	-5	-4	19
2	2	-8	-3	-1	5	19
3	0	7	-2	3	-9	21
4	-1	2	-5	5	-1	14
5	-5	-6	-1	3	-1	16
6	-3	3	4	0	-4	14
7	-2	-1	0	-5	3	11
8	-10	10	-4	-5	2	31
9	3	0	2	2	-4	11
10	4	-9	7	-6	-3	29

Table 5 shows all the weights in this network, rounded to integers. The columns correspond to the categories, and the rows correspond to the features. The bias inputs to the category units are also shown as the first row, as are the sums of the absolute values of the weights in each row. Looking at the column labeled NP, for example, it is clear that the first, ninth, and tenth features strongly code for NP, while the eighth and fifth features code against NP. Looking at the column labeled VP, the third and eight features code for it, and the second and tenth against.

The "strength" of each row indicates how categorical or distinctive a feature is. The tenth feature, for example, strongly codes for NP and PP and against VP, AP, and S. The features which do not connect strongly everywhere, like the seventh and ninth, are used for discriminations within the categories. With regard to the binary-versus-real question raised earlier, it seems that RAAM may build a hybrid code. Strong binary distinctions are used for categorical judgements, while weaker analog distinctions are used for discriminating (and labeling) members within the categories.

4.2.1. Geometric Interpretation

An alternative means of understanding these representations may come from geometry. The terminal patterns are vertices of a *k*-dimensional hypercube which contains all of the non-terminal patterns.

For binary trees, a RAAM is finding a consistent invertible mapping which works the same way on composable pairs of vertices, as it does on the internal points that are also composed. To view an image of this, a 6-3-6 RAAM was trained on the two trees ((A B)(C D)) and ((A C)(B D)), with A = (0 0 0), B = (1 0 0), C = (0 1 0), and D = (1 1 0); i.e. with A, B, C, and D the four points on the "floor" of a 3-D cube.

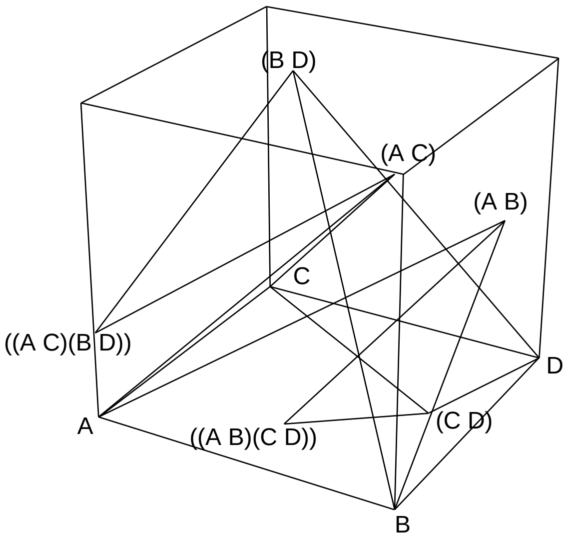

Figure 12. *Perspective diagram for the 3-dimensional codes developed for the trees ((A B)(C D)) and ((A C)(B D)).*

Figure 12 shows a perspective plot of the 3-dimensional hypercube for the codes developed for these two trees. If one stares long enough, taking each pair of composable points in one's mental left and right hands, one can see triangles falling forward as they reduce in scale.

Saund [34] has investigated (non-recursive) auto-association as a method of *dimensionality reduction*, and asserted that, in order to work, the map must be constrained to form a small dimensional parametric surface in the larger dimensional space. Consider just a 2-1-2 auto-associator. It is really an invertible mapping from certain points on the unit square to points on the unit line. In order to work, the network might develop a parametric 1-dimensional curve in 2-space, perhaps a set of connected splines. As more and more points need to be encoded, this parametric curve must get ‘‘curvier’’ to cover them. In the limit, especially if there are any dense "patches" of 2-space which need to be covered, it can no longer be a 1-dimensional curve, but must become a space-filling curve with a fractal dimension [35]. The notions of associative and reconstructive memories with fractal dimensions are further discussed elsewhere [36].

4.3. Applications

4.3.1. Associative Inference

Since RAAM can devise representations of trees as numeric vectors which then can be attacked with the fixed-width techniques of neural networks, this work might lead to very fast inference and structural transformation engines. The question, of course, is whether the patterns for trees can be operated on, in a systematic fashion, without being decoded first. Below is a very simple demonstration of this possibility.

Since the RAAM for the propositional triples was able to represent all 16 cases of (LOVED X Y), it should be possible to build an associative network which could perform the simple implication: "If (LOVED X Y) then (LOVED Y X)". This would be a trivial shifting task if performed on an explicit concatenative representation. However, since the (48 bit) triples are compressed into 16-dimensional pattern vectors, it is not quite as simple a job.

The task is to find an associator which can transform the compressed representation for each antecedent (e.g. (LOVED MARY JOHN)) into the compressed representation for its consequent (e.g. (LOVED JOHN MARY)). Using back-propagation, a 16-8-16 feed-forward network was trained on 12 of the 16 pairs of patterns (to within 5% tolerance) and was then able to successfully transform the remaining 4 pairs.

What about a system which would need to follow long chains of such implications? There has recently been some work showing that under certain conditions, feed-forward networks with hidden layers can compute arbitrary non-linear mappings [37-39]. Therefore, I anticipate that the sequential application of associative inference will be able to be compiled, at least by slow training, into fast networks of few layers.

Consider homogenous coordinate transformations (in computer graphics), where the linear nature of the primitive operations (scaling, rotation, and translation) allows any sequence of them to be "compiled" into a single matrix multiplication. The field of AI has not, to date, produced any compiling methods which can rival this speedup, because most interesting AI problems are nonlinear and most interesting AI representations are not numeric. The point is that given suitable representations, efficient non-linear mapping engines could generate significant speed improvements for inferential processing.

4.3.2. Massively Parallel Parsing, Revisited

I introduced this paper by noting that natural language processing posed some problems for connectionism, precisely because of the representational adequacy problem. One cannot build either a parser or a generator without first having good "internal" representations. RAAMs can devise these compositional representations, as shown by the experiment on semantic triples, which can then be used as the target patterns for recurrent networks which accept sequences of words as input.

A feasibility study of this concept has been performed as well, using a sequential cascaded network [40], a higher-order network with a more restricted topology than Sigma-Pi [41]. Basically, a cascaded network consists of two subnetworks: The *function network* is an ordinary feed-forward network, but its weights are dynamically computed by the purely linear *context network*, whose outputs determine each weight of the function net. In a sequential cascaded network, the outputs of the function network are directly fed back to the inputs of the context network. This network is trained with presentations of initial context, input sequences, and desired final state.

Table 6. *10-bit input patterns for connectionist parser.*

WORD	CLASS	IDENTITY
JOHN	1 0 0 0 0	1 1 0 0 0
MAN	1 0 0 0 0	0 1 0 0 0
PAT	1 0 0 0 0	1 1 1 0 0
MARY	1 0 0 0 0	1 0 1 0 0
HE/HER	1 0 0 0 0	0 1 0 1 0
TELESCOPE	0 1 0 0 0	0 0 1 0 1
SPAGHETTI	0 1 0 0 0	1 0 0 1 0
CHOPSTICKS	0 1 0 0 0	0 0 1 1 0
HILL	0 1 0 0 0	0 1 0 0 0
MEAT	0 1 0 0 0	1 0 0 0 1
ON	0 0 1 0 0	1 0 0 0 0
WITH	0 0 1 0 0	0 1 0 0 0
WHO	0 0 1 0 0	0 0 1 0 0
BY	0 0 1 0 0	0 0 0 1 0
ATE	0 0 0 1 0	0 0 1 0 0
HIT	0 0 0 1 0	0 0 0 1 0
SAW	0 0 0 1 0	0 0 0 0 1
LOVED	0 0 0 1 0	0 0 0 1 1
HOPED	0 0 0 1 0	0 1 1 0 0
THOUGHT	0 0 0 1 0	0 1 0 1 0
KNEW	0 0 0 1 0	0 1 0 0 1
LONG	0 0 0 0 1	0 0 0 1 0
SHORT	0 0 0 0 1	0 0 0 0 1

A new 10-bit similarity-based encoding was created for the words appearing in the sentences, making HE and HER identical. The first 5 bits define the class, and the second 5 bits distinguish the members. The patterns are displayed in Table 6. A sequential cascaded network consisting of a 10-10-16 function network and a 16-286 context network was trained using sequences of these bit patterns corresponding to the sentences in Table 1. The initial context vectors were all zeroes, and the desired final states were the compressed 16-dimensional representations devised by the 48-16-48 RAAM for the trees in Table 3 (not including 10b).

This system is the closest thing yet to a barely adequate connectionist system for processing language: Given a variable-length sequence of words, the network returns, in linear time, a 16-dimensional vector, which can be decoded into a "meaning" by a RAAM, and can perhaps be operated upon by associative inference engines.

On the one hand, this system has extreme deficiencies if it is evaluated as a cognitive model. It can only produce a single tree for a sentence, and only handles a very small corpus of sentences. The simplifying assumption, that internal representations can first be devised and then used as target patterns, is questionable. On the other, the system has some very interesting aspects. Besides the fact that it runs in linear time and outputs a compositional representation for the sentences, it automatically performs prepositional phrase attachment (i.e., correctly parses the ‘‘MARY ATE SPAGHETTI WITH MEAT/CHOPSTICKS’’ examples) and pronoun resolution (i.e., automatically replaces HE or HER with the proper filler). Finally, it is the first connectionist parser which can deal with embedded structures without resorting to external symbolic computational power.

4.4. Further Work

There is a great deal of research still to be conducted in this area, besides the conversion of the small feasibility studies into both falsifiable cognitive models and reliably engineered artifacts. Immediate concerns include:

- Understanding the convergence and stability properties of the "moving target" learning strategy; both empirical and analytical studies are called for. Similarly, the relationship between the termination condition (using τ an ν) and the depth capacity of RAAM needs to be better understood..
- Developing a complete understanding of the representations and mechanisms which are developed. A good outcome would be a general representational scheme which could be analytically derived for a particular representational task without relying on slow, gradient-descent learning.

5. Conclusion

Here is a conundrum for theories of human and machine learning: *Which came first*, the mental procedure or the mental representation? Minsky and Papert claimed that the representational egg must come before the procedural chicken, while Fodor and Pylyshyn claimed to intimately know the egg and, by extension, the exclusive class of fertile chickens. The flip side, of course, is that this perfect egg may only be layable by an impossible chicken: A formal representational theory, specified without consideration of its own genesis, may not be learnable by any mechanism in principle.

This work points to biologically certified way out of the dilemma: **Co-Evolution.** The representations and their associated procedures develop slowly, responding to each other’s constraints through a changing environment. The constraint that the representations fit into fixed-width patterns interacts with the constraint that the patterns must compose in certain well-formed ways, giving rise to fixed-width patterns which capture structural similarity in spatial distance.

The RAAM architecture has been inspired by two powerful ideas. The first is due to Hinton [42], who showed that, when properly constrained, a connectionist network can develop semantically interpretable representations on its hidden units. The second is an old idea, that given a sufficiently powerful form of learning, a machine can learn to efficiently perform a task by example, rather than by design. Taken together, these ideas suggest that, given a task, specified by example, which *requires* embedded representations, a network might be able to develop these representations itself.

It turns out that there is no *single* task which requires such representations. There have to be at least two tasks; one to construct the representations, and another to access them. On address-based machines, these tasks, such as string concatenation and array indexing, are so computationally primitive and natural that they fall far below notice. They are not natural to neural networks and thus need to be examined anew. Here, the resulting task-specific mechanisms, the compressor and reconstructor, together form a *reconstructive* memory system, in which only traces of the actual memory contents are stored, and reliable facsimiles are created with the use of domain knowledge.

The systematic patterns developed by RAAM are a very new kind of representation, a **recursive, distributed representation**, which seems to instantiate Hinton's notion of the "reduced description" mentioned earlier [19]. They combine apparently immiscible aspects of well-understood representations: They act both like feature vectors with their fixed width and simple measures of similarity, and like pointers, so that, with simple efficient procedures their contents can be "fetched." Even further, they act like compositional symbol structures: Simple associative procedures, such as the reconstructor, pattern classifiers, and pattern transformers, are clearly sensitive to their internal structure.

However, unlike feature vectors, these representations recursively combine into constituent structures, according to statistically inferred well-formedness constraints. Unlike pointers (or symbols like G0007), they contain information suitable for similarity measurements and, thus, nearest-neighbor judgements. And, unlike symbol structures, they can be easily compared, and do not have to be taken apart in order to be worked on. Recursive distributed representations may thus lead to a reintegration of the syntax and semantics at a very low level.

Currently, symbolic systems use information-free "atoms" which physically combine (through bit or pointer concatenation) in a completely unrestricted fashion. Thus, for any domain, a syntax is required to restrict those "molecules" *after the fact*, to the set of semantically interpretable ones. With further work, recursive distributed representations might undergo a metamorphism into symbols which contain their own meanings and physically combine *only* in a systematic fashion. After all, real atoms and molecules do so all the time.

Acknowledgements

This work has been partially supported by the the State Legislatures of New Mexico and Ohio and by the Office of Naval Research, under grant N00014-89-J-1200. Thanks to Tony Plate, who implemented back-propagation in C, and to Yoshiro Miyata for a copy of his hierarchal cluster program. Comments from B. Chandrasekaran, G. Hinton, J. McClelland, D. Touretzky, T. VanGelder, and many, many others helped to improve this presentation.

References

1. G. W. Cottrell, Connectionist Parsing, *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, Irvine, CA, 1985.
2. M. Fanty, Context-free parsing in Connectionist Networks, TR174, University of Rochester, Computer Science Department, Rochester, N.Y., 1985.
3. B. Selman, Rule-Based Processing in a Connectionist System for Natural Language Understanding, CSRI-168, University of Toronto, Computer Systems Research Institute, Toronto, Canada, 1985.
4. S. J. Hanson and J. Kegl, PARSNIP: A connectionist network that learns natural language grammar from exposure to natural language sentences, *Proceedings of the Ninth Conference of the Cognitive Science Society*, Seattle, 1987, 106-119.
5. J. McClelland and A. Kawamoto, Mechanisms of Sentence Processing: Assigning Roles to Constituents, in *Parallel Distributed Processing: Experiments in the Microstructure of Cognition*, vol. 2, J. L. McClelland, D. E. Rumelhart and the PDP research Group (ed.), MIT Press, Cambridge, 1986.
6. J. B. Pollack and D. L. Waltz, Natural Language Processing Using Spreading Activation and Lateral Inhibition, *Proceedings of the Fourth Annual Cognitive Science Conference*, Ann Arbor, MI, 1982, 50-53.
7. D. L. Waltz and J. B. Pollack, Massively Parallel Parsing: A strongly interactive model of Natural Language Interpretation, *Cognitive Science 9*, 1 (1985), 51-74.
8. W. G. Lehnert, Case-based problem-solving with a large knowledge base of learned cases, *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, 1987, 301-306.
9. G. Berg, A Parallel Natural Language Processing Architecture with Distributed Control, *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Seattle, 1987, 487-495.
10. M. Minsky and S. Papert, *Perceptrons*, MIT Press, Cambridge, MA, 1988.

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

12. D. E. Rumelhart, G. Hinton and R. Williams, Learning Internal Representations through Error Propagation, in *Parallel Distributed Processing: Experiments in the Microstructure of Cognition*, vol. 1, D. E. Rumelhart, J. L. McClelland and the PDP research Group (ed.), MIT Press, Cambridge, 1986, 25-40.

13. J. L. McClelland and D. E. Rumelhart, An interactive activation model of the effect of context in perception: Part 1. An account of basic findings, *Psychology Review 88*, (1981), 375-407.

14. R. Allen, Several Studies on Natural Language and Back Propagation, *Institute of Electrical and Electronics Engineers First International Conference on Neural Networks*, San Diego, 1987, II-335-342.

15. T. J. Sejnowski and C. R. Rosenberg, Parallel Networks that Learn to Pronounce English Text, *Complex Systems 1*, (1987), 145-168.

16. E. Charniak and E. Santos, A context-free connectionist parser which is not connectionist, but then it is not really context-free either., in *Advances in Connectionist & Neural Computation Theory*, J. Barnden and J. Pollack (ed.), Ablex, Norwood, NJ, 1989.

17. G. E. Hinton, Distributed Representations, CMU-CS-84-157, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, 1984.

18. A. H. Kawamoto, Dynamic Processes in the (Re)Solution of Lexical Ambiguity, Doctoral Dissertation, Department of Psychology, Brown University, Providence, 1985.

19. G. Hinton, Representing Part-Whole hierarchies in connectionist networks, *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, Montreal, 1988, 48-54.

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

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

22. D. S. Touretzky, Representing and transforming recursive objects in a neural network, or ‘‘trees do grow on Boltzmann machines’’, *Proceedings of the 1986 Institute of Electrical and Electronics Engineers International Conference on Systems, Man, and Cybernetics*, Atlanta, GA, 1986.

23. D. E. Rumelhart and J. L. McClelland, On Learning the Past Tenses of English Verbs, in *Parallel Distributed Processing: Experiments in the Microstructure of Cognition*, vol. 2, J. L. McClelland, D. E. Rumelhart and the PDP research Group (ed.), MIT Press, Cambridge, 1986, 216-271.

24. S. Pinker and A. Prince, On Language and Connectionism: Analysis of a parallel distributed processing model of language inquisition., *Cognition 28*, (1988), 73-193.

25. M. Mozer, Inductive information retrieval using parallel distributed computation, Technical Report, Institute for Cognitive Science, UCSD, La Jolla, 1984.

26. R. Rosenfeld and D. Touretzky, Four capacity models for coarse-coded symbol memories, *Complex Systems 2*, (1988), 463-484.

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

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

29. J. L. Elman, Finding Structure in Time, Report 8801, Center for Research in Language, UCSD, San Diego, 1988.

30. R. Miikkulainen and D. G. Dyer, Forming Global Representations with Back Propagation, *Proceedings of the Institute of Electrical and Electronics Engineers Second Annual International*

Conference on Neural Networks, San Diego, 1988.

31. D. C. Plaut, S. Nowlan and G. E. Hinton, Experiments on learning by back-propagation, CMU-CS-86-126, Computer Science Dept., Carnegie Mellon University, Pittsburgh, 1986.

32. J. H. Holland, K. J. Holyoak, R. E. Nisbett and P. R. Thagard, *Induction: Processes of Inference, Learning, and Discovery*, MIT Press, Cambridge, 1986.

33. M. I. Jordan, Serial Order: A Parallel Distributed Processing Approach, ICS report 8608, Institute for Cognitive Science, UCSD, La Jolla, 1986.

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

35. B. Mandelbrot, *The Fractal Geometry of Nature*, Freeman, San Francisco, 1982.

36. J. B. Pollack, Implications of Recursive Distributed Representations, in *Advances in Neural Information Processing Systems*, D. Touretzky (ed.), Morgan Kaufman, Los Gatos, CA, 1989.

37. K. Hornik, M. Stinchcombe and H. White, Multi-layer Feedforward Networks are Universal Approximators, *Neural Networks*, To Appear.

38. R. P. Lippman, An introduction to computing with neural networks, *Institute of Electrical and Electronics Engineers ASSP Magazine April*, (1987), 4-22.

39. A. S. Lapedes and R. M. Farber, How Neural Nets Work, LAUR-88-418, Los Alamos, 1988.

40. J. B. Pollack, Cascaded Back Propagation on Dynamic Connectionist Networks, *Proceedings of the Ninth Conference of the Cognitive Science Society*, Seattle, 1987, 391-404.

41. R. Williams, The Logic of Activation Functions, in *Parallel Distributed Processing: Experiments in the Microstructure of Cognition*, vol. 1, D. E. Rumelhart, J. L. McClelland and the PDP research Group (ed.), MIT Press, Cambridge, 1986, 423-443.

42. G. E. Hinton, Learning Distributed Representations of Concepts, *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, Amherst, MA, 1986, 1-12.