Exploring the Symbolic/Subsymbolic Continuum: A Case Study of RAAM*

Douglas S. Blank (blank@iuvax.cs.indiana.edu)
Lisa A. Meeden (meeden@iuvax.cs.indiana.edu)
James B. Marshall (marshall@iuvax.cs.indiana.edu)

Department of Computer Science
Indiana University
Bloomington, Indiana 47405

May 6, 1991

1 Introduction

It is difficult to clearly define the symbolic and subsymbolic paradigms; each is usually described by its tendencies rather than any one definitive property. Symbolic processing is generally characterized by hard-coded, explicit rules operating on discrete, static tokens, while subsymbolic processing is associated with learned, fuzzy constraints affecting continuous, distributed representations. In addition, programming languages such as Lisp and mechanisms such as Turing machines are typically associated with the symbolic paradigm, while connectionism is frequently associated with the subsymbolic paradigm. Debates contrasting the two paradigms sometimes center on these mechanisms, for example comparing the capabilities of Turing machines with those of connectionist networks (see Adams, Aizawa, and Fuller in this volume). However, connectionist networks can be proven to be computationally equivalent to the abstract notion of Turing machines [Franklin and Garzon, 1990]. Therefore the computational mechanism is not the crucial issue in separating the symbolic and subsymbolic paradigms. What then is the crucial issue?

We believe there are three major issues which distinguish the symbolic paradigm from the subsymbolic paradigm: (1) the type of representations; (2) the style of composition; and (3) the functional characteristics. We have summarized the key elements of these differences between the two paradigms in Table 1. However, most cognitive science and classical artificial intelligence (AI) models cannot be completely characterized as either purely symbolic or purely subsymbolic using these criteria. Instead, most models fall somewhere in between the two extremes, or in the so-called "Gap." For this reason, it seems appropriate to view the paradigms as defining two opposite corners of a three-dimensional continuum as shown in

*To appear in *Closing the Gap: Symbolism vs. Connectionism*, J. Dinsmore, editor. 1992, Lawrence Erlbaum Associates.

	Subsymbolic	**Symbolic**
Representation	distributed	atomic
	continuous	discrete
	emergent	static
	use affects form	arbitrary
Composition	superimposed	concatenated
	context-sensitive	systematic
Functionality	microsemantic	macrosemantic
	holistic	atomistic

Table 1: Comparison of the Subsymbolic and Symbolic Paradigms

Figure 1.[1] In the following introductory sections we examine each of the issues from Table 1 in detail and then discuss where to place some existing models within this symbolic/subsymbolic continuum.

1.1 Representation

As Smolensky has noted, the term *subsymbolic paradigm* is intended to suggest symbolic representations which are built out of many smaller constituents: "entities that are typically represented in the symbolic paradigm by symbols are typically represented in the subsymbolic paradigm by a large number of subsymbols." Smolensky suggests that for the purposes of relating these two paradigms, it is often important to analyze subsymbolic models at a higher level: "to amalgamate, so to speak, the subsymbols into symbols." [2] One problem with this type of analysis is that a conglomerate of subsymbols does not form a traditional symbol. Classically, symbols have been *arbitrary* labels, such as strings of letters, which are *atomic*, *discrete*, and *static*. In contrast, a symbol in the subsymbolic paradigm is *distributed* over a collection of subsymbols, and each subsymbol may be associated with *continuous* numerical values. In addition, the subsymbolic paradigm is strongly committed to learning at the subsymbolic level. Through learning, an amalgamated symbol gradually *emerges* in such a way that its *form reflects its function*, or use, in the training tasks. We will see examples of this in the experiments described in Section 5.

Typical symbols in a symbolic model might be the letter strings *waiter* or *customer*. These symbols may be placed into structured relationships with other symbols, and may be bound to a variety of values during the course of processing, but the forms of the symbols themselves never change. The symbolic model would work equally well if *waiter* were replaced by *xyz* throughout the model's data structures, since *waiter* is simply an atomic label possessing no internal structure of its own. In contrast, a typical symbol in a subsymbolic model might be the pattern of continuous values *[+0.562 -0.891 -0.143 -0.382 +0.966]*. During processing, this might evolve to the slightly different but similar pattern *[+0.589 -0.900 -0.139 -0.412*

[1]Table 1 and Figure 1 were inspired by Robert Port and Timothy van Gelder. They conceived of viewing classical and connectionist models as varying along a number of abstract dimensions which define a space of possible representational schemes. They were specifically interested in representations for natural language [Port and van Gelder, 1991]. We have extended these ideas by contrasting paradigms in general rather than focusing solely on representational issues.

[2][Smolensky, 1988, page 3]

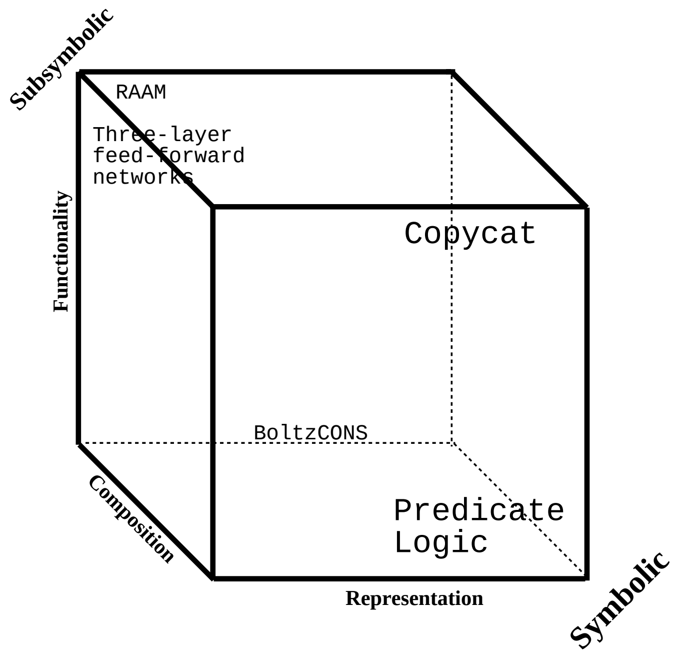

Figure 1: **Subsymbolic/symbolic continuum:** This figure illustrates the three dimensional space of paradigms defined by the three characteristics from Table 1: representation (across the bottom), composition (extending out), and functionality (up the side). The Gap between the symbolic (front, lower, right corner) and subsymbolic (back, upper, left corner) paradigms can be viewed as the central region of the cube. Five models are shown in their approximate position in the space. Font size reflects distance from the front of the box.

+0.999], which, although distinct from the original symbol, still behaves in a closely related way. However, the subsymbolic model would produce very different results if this symbol were replaced by some other completely arbitrary pattern.

These fundamental differences in representation between the two paradigms directly influence the functional capacities and compositional styles exhibited by each.

1.2 Composition

One of the main criticisms that has been leveled against the subsymbolic approach is that the representations developed by subsymbolic models are unable to exhibit useful compositional structure [Fodor and Pylyshyn, 1988]. It is claimed that the representations operated on by subsymbolic computation cannot be combined into higher-order composite representations which preserve the integrity of the constituent parts out of which they are assembled. This capability is clearly present in symbolic models. One need only imagine a model which operates on representations encoded as symbols organized into structured trees or lists. These representations can be easily decomposed into their constituent parts by using standard destructuring operations, such as the *car* or *cdr* operators in Lisp, and then reassembled into new representations using standard *concatenative* operations, such as the *cons* operator in Lisp.[3]

Pollack has developed a subsymbolic model, called Recursive Auto-Associative Memory or RAAM, that provides a direct counterexample to the claims that subsymbolic models cannot exhibit useful compositional structure [Pollack, 1988, Chalmers, 1990b]. This chapter will focus on the RAAM model as an exemplar of the subsymbolic paradigm. However, we do not wish to claim that RAAM is an accurate model of human cognition; we simply feel that it provides a clear illustration of the characteristics of the subsymbolic region of the continuum.

The basic purpose of RAAM is to allow familiar recursive data structures such as trees and lists to be encoded into distributed representations suitable for processing by connectionist networks. However, the result of subsymbolic composition is very different from the explicit concatenative compositional structures created in the symbolic paradigm. A RAAM produces fixed-length distributed patterns of continuous numerical values which encode compositional structure implicitly. Since the length of the distributed patterns cannot expand or contract to match the size and depth of the compositional structures to be represented (like a Lisp list does as elements are *cons*ed onto it), the structure must be *superimposed* across the fixed subsymbols. In this way, a composite of symbols in RAAM is itself a symbol, and has the same general form and functionality as the original symbols it was constructed from. The details of how a RAAM constructs and decomposes compositional structures is a major focus of this chapter and will be examined extensively in subsequent sections.

When symbols are composed in a symbolic style, they form well-defined structures with a *systematic* organization. This explicit structuring allows for precise relationships between symbolic symbols. For instance, the form of *waiter* in a symbol structure will not be altered by using that symbol in two slightly different ways, such as *(he gave it to the waiter)* and *(she handed it to a waiter)*. On the other hand, the subsymbolic paradigm allows for context

[3]Given that x is bound to the list (1 2 3) in Lisp, *(car x)* returns 1, *(cdr x)* returns (2 3), and *(cons 4 x)* returns (4 1 2 3).

to play a role in a compositional representation. In a subsymbolic system, the same symbol used in slightly different contexts may reflect that difference in its form. Although some systematicity is useful, too much can make a system inflexible. This *context-sensitivity* may allow for the functionality of a subsymbolic system to take advantage of information which has been abstracted out of the symbols in a symbolic system.

1.3 Functionality

The functionality of the two paradigms depends directly on their style of representations. Subsymbolic representations reflect the tasks encountered in the training process, in that the generalizations needed to successfully perform the tasks tend to be captured within the internal structure of the representations themselves. In this way, symbols used similarly will develop similar representations. This gives subsymbolic symbols internal relationships to one another, or a *microsemantics*. In contrast, symbolic symbols are arbitrary and atomic, and have no internal semantics. The complete functionality of symbolic systems rests on the structured relationships existing between the arbitrary symbols, or a *macrosemantics* [Dyer, 1990]. In effect, a microsemantics is internal to symbols and a macrosemantics is external to symbols.

Probably the most important difference in the functionality of the two paradigms lies in the methods by which symbols can be operated upon. If a composite structure in a symbolic system, say a Lisp list, were to have its fourth element tested for some criterion, one must first remove the first three elements of the list to get to the fourth. In fact, to do anything to the list involving the elements of that list, one must first decompose it. Thus, symbolic list structures can only be operated on *atomistically*. Since many AI practitioners have exclusively used concatenative data structures, the need for this initial deconstruction step seems to be a natural consequence of building data structures. In the subsymbolic paradigm, however, an operation can act *holistically* on an entire symbol structure [Blank, 1990, Chalmers, 1990a]. In this way, a subsymbolic operation can, in one step, perform a complex function without decomposing the representation of a symbol structure into its constituent parts. A number of examples of holistic operations on composite RAAM representations will be described later in this chapter.

1.4 The Symbolic/Subsymbolic Continuum

Figure 1 depicts the symbolic/subsymbolic continuum as a three-dimensional space delineated by the three main issues given in Table 1. Positioned within this continuum are several representative models selected to illustrate different aspects of the continuum: formal logical inference systems which use predicate calculus; three-layer feed-forward connectionist networks; BoltzCONS, a constraint-satisfaction memory system; Copycat, an analogy-making system; and the RAAM model to be discussed in this chapter. The next subsections will briefly discuss these systems and their positions within this continuum.

1.4.1 Predicate Logic

In the predicate calculus, abstract atomic objects are called *tokens* and can be concatenated together in recursively defined ways to form arbitrarily complicated logical expressions. These tokens and structured collections of tokens can be used to represent knowledge by assigning to them some type of semantic interpretation, and specifying transformation rules which manipulate them in ways consistent with this interpretation. These expressions and transformation rules can be said to model particular aspects of the world to the extent that the causal relationships which exist among concepts and information in the outside world are captured by the causal interaction of the tokens and logical expressions as governed by the rules.

Certain tokens are assigned unique interpretations, perhaps as variables or logical operators such as And, $ForAll$, or $\rightarrow$ (implication). We could perhaps represent the fact that all dogs are mammals by the expression:

$$[ForAll(x)\,[Dog(x) \rightarrow Mammal(x)]]$$

and the fact that Fido is a dog by the expression $Dog(Fido)$. These representations can then be manipulated by transformation rules in order to extract useful information from them. Given the fact $Dog(Fido)$ and the previous implication expression, an inference system might deduce the fact $Mammal(Fido)$ by first breaking these expressions up into their constituent tokens and then recombining them using logical transformation rules such as unification and resolution.

The inference process is atomistic and based entirely on the macrosemantics of the tokens. The database facts are represented concatenatively and the inference rules are applied systematically to these facts. The tokens are static, discrete, and arbitrary. Therefore standard inference systems based on predicate calculus, such as described above, are prototypical examples of the symbolic paradigm and have been positioned in the symbolic corner of the continuum in Figure 1.

1.4.2 Three-layer Feed-forward Networks

An example of a connectionist subsymbolic architecture is a network of processing units arranged into hierarchical layers. These layers are usually shown with input coming in from the bottom and output exiting from the top (see Figure 2). Each unit in any given layer (except the output layer) is connected by weighted connections to each unit in the layer above it. Various amounts of activation are applied to each of the units in the bottom layer, representing some particular pattern being presented as input to the network. This activation then flows across the connections to higher layers of the network, with the weights on the connections mediating the amount of activation that is passed on to successive units. The final pattern of activation present on the topmost layer is considered to be the output pattern produced by the network from the given input pattern.

A learning algorithm such as back-propagation [Rumelhart et al., 1986] can be repeatedly applied to the network, enabling it to learn to associate arbitrary pairs of input and output patterns by gradually adjusting the weights on the connections between units. These input/output patterns can be interpreted as representing information received from and sent to the network's surrounding environment. As a result of this training process, the network learns to recode each of the input patterns into different patterns of activation at each successive intermediate layer of units (called hidden layers), so that the appropriate output pattern may be successfully generated at the output layer. This process of learning to recode input

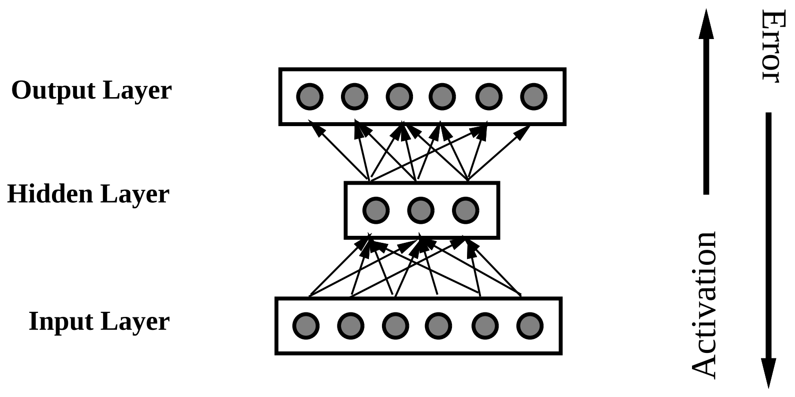

Figure 2: **Basic connectionist network:** This schematic diagram of a simple network depicts layers as rectangles, units as shaded circles, and weights as arrows. In this standard feed-forward three-layer network, activation begins at the bottom in the Input Layer, proceeds upwards through the Hidden Layer, and ends in the Output Layer. Learning is accomplished by propagating errors from the Output Layer back through the network to the Input Layer.

patterns into intermediate patterns of activation spread across the hidden layers amounts to the development of distributed internal representations of the input information by the network itself.

The ability of connectionist models to develop their own distributed internal representations—or *hidden representations*—is an extremely important property of this class of models, and provides the basis for placing them in the subsymbolic corner of Figure 1. Individual hidden units participate in the simultaneous creation of many different hidden representations, which are superimposed on top of each other in the network, since different hidden representations are activated across the same set of hidden units depending on the current input to the network. In this way individual units can be viewed as playing different roles, depending on which hidden representation is currently active.

1.4.3 BoltzCONS

BoltzCONS [Touretzky, 1990] is a system which stores and processes traditional symbolic data structures such as lists and trees in a nontraditional fashion (see also Barnden in this volume). The central component of BoltzCONS is a distributed memory which allows Lisp-like data structures to be stored and retrieved. After storing a complex nested list structure (i.e. a tree) in memory, the tree may be traversed by performing associative retrieval on the distributed memory. The memory is implemented as a constraint-satisfaction connectionist network containing a large number of units, each of which is configured to respond to a randomly-specified subset of all possible trees. The presence or absence of a specific tree in memory is determined by how many of the tree's associated units are activated. Storing a tree in memory is accomplished by activating all of the tree's associated units; similarly, deleting a tree is accomplished by turning off all of the units. Over time, due to the distributed nature of the representations, storing and deleting from the memory causes a gradual degradation of the memory's contents.

In terms of Table 1, BoltzCONS's representations could be considered subsymbolic. The symbol for a tree is distributed across many units in the memory and may be superimposed over other symbols, as in many connectionist models. At any given moment, trees are present in memory to some degree, depending on the total activation across their associated units, rather than always being either completely present or absent. Thus, symbols in BoltzCONS are more continuous than they are discrete and the ability to accurately retrieve them depends on the current contents of the memory. On the other hand, these representations contain no microsemantics; they are arbitrary and static. The nodes in the memory which will respond to a given set of trees are fixed initially and remain bound to the same trees throughout the processing. As in many symbolic systems, the representations in BoltzCONS rely on the macrosemantics between the symbols for their functionality.

BoltzCONS combines aspects of both symbolic and subsymbolic paradigms, but does not seem to be completely characterizable as either. Its representations are continuous and distributed but are also static and arbitrary. For this reason it has been positioned in the middle of the representation dimension of the continuum. Its composition is context-sensitive and superimposed, positioning it on the subsymbolic end of the composition dimension of the continuum. Finally its functionality depends on macrosemantics and therefore it has been positioned at the symbolic end of the functionality dimension.

1.4.4 Copycat

Another system which lies in the Gap is Copycat, a program designed to solve idealized analogy problems [Hofstadter, 1984, Hofstadter and Mitchell, 1991]. These analogies are stated in terms of letter strings: given that *letter-string1* changes into *letter-string2*, what does *letter-string3* change into? A typical problem might be: "if $abc \rightarrow abd$ then $ijk \rightarrow ?$"

When presented with a particular analogy problem, Copycat gradually builds an internal representation of the problem in terms of a set of basic concepts intrinsic to the program. Constructing this representation entails building a mapping between corresponding pieces of the problem, which then serves as a guide for producing a reasonably analogous answer. Building the mapping is accomplished over time by the collective efforts of a large number of independent, small, locally-acting processes known as *codelets* executing in parallel. Individual codelets are responsible for building only small pieces of the overall mapping structure. At any given time there may be several incompatible pieces of structure competing for inclusion in the final mapping, in which case the winner is generally chosen according to the degree to which it strengthens the consistency of the context in the already-existing structures. The quality of the final answer produced by Copycat directly depends on the construction of a strong final mapping, and the central task of the program is essentially to search for such a mapping through a vast space of possibilities.

However, there is no overarching executive process monitoring the construction of the mapping, or controlling the activity of the codelets. The global course of the processing—the search for a good characterization of the problem through the space of all possible mappings—emerges at a higher level, out of the sustained activities of many hundreds of codelets. Thus, Copycat has a strong subsymbolic flavor, at least along the functional dimension.

On another dimension, the representations developed by Copycat, namely the mappings between letter strings constructed by the codelets, retain many of the qualities of traditional symbolic representations. Structural components included in the final mapping are either present or absent, although it is possible for representational structures to be only tentatively present during the course of building the mapping. These virtual structures may eventually acquire a permanent status in the final mapping, or they may be replaced by stronger structures. The notion of partially-present representational components exists in Copycat despite the symbolic nature of its representations, although to a lesser degree than in BoltzCONS. Therefore Copycat has been placed in the upper, front, right corner in Figure 1.

In this section, we have positioned several models within the symbolic/subsymbolic continuum. In the remainder of the chapter we will examine the RAAM model in terms of the three dimensions in Figure 1 and demonstrate that it belongs in the subsymbolic corner of the continuum.

2 Recursive Auto-Associative Memory

Recall that RAAM is designed to allow traditional symbolic data structures such as trees to be represented subsymbolically as distributed patterns of activation. The general architecture of RAAM is a three-layer feed-forward network of processing units, in which the input and output layers contain equal numbers of units to allow for *auto-association* of the patterns presented to the network. That is, given some pattern on the input layer, the network must reproduce that same pattern on the output layer by first *encoding* the input into some internal representation distributed across the hidden layer, and then *decoding* this hidden representation back to the original pattern on the output layer. The back-propagation learning algorithm is used to perform the auto-association. Furthermore, the hidden layer must be smaller than the input and output layers in order to force the network to accomplish the auto-associative mapping by creating compressed hidden representations. The set of connections from the input layer to the hidden layer serve as the encoding (or composing) mechanism, and the set of connections from the hidden layer to the output layer serve as the decoding (or decomposing) mechanism.

Using the auto-association technique, a RAAM can encode general tree structures of variable depth and fixed branching size into fixed-length distributed representations. The depth may be arbitrary in that no specific upper bound is placed on the depth of the trees encoded, just as there is no specific upper bound on the number of trees which may be stored in a single RAAM network. Of course, as with any connectionist model, the number of patterns which may be stored and retrieved accurately by any particular RAAM depends on the network's size and the number of training patterns involved. But in principle, the RAAM model is capable of encoding arbitrarily large recursive data structures into distributed, compositional representations, and then recovering the underlying constituent structures out of which these representations are composed.

Consider a tree to be represented in a RAAM as being structured in the following manner: the leaves of the tree correspond to individual elements stored in the tree, and the internal branching nodes specify the way these elements are related in the tree. Suppose we represent each of the leaf elements by some unique pattern of activation of length n. The particular encoding we choose for the elements is arbitrary, but the chosen patterns must all be distinct. A RAAM for encoding a set of trees with fixed branching size k will consist of input and

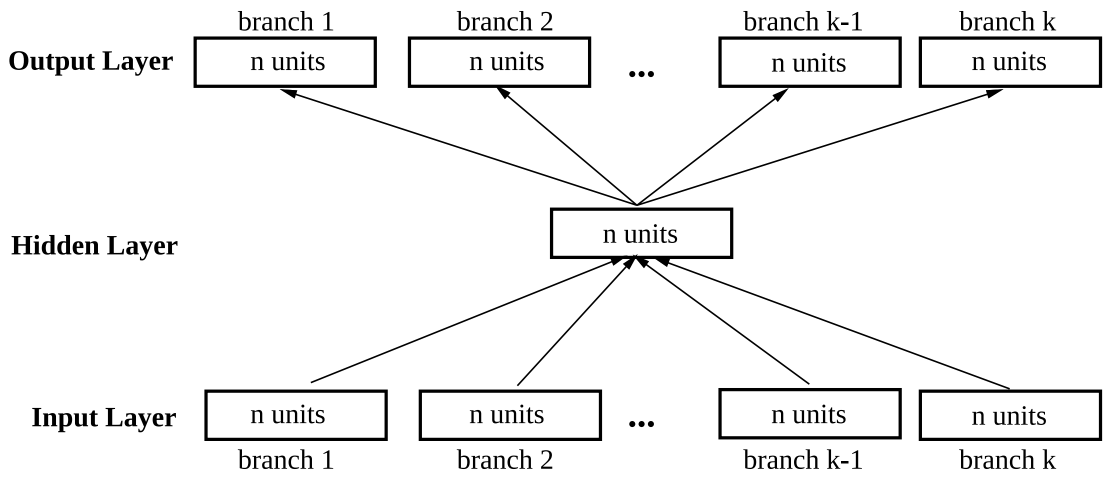

Figure 3: **General RAAM architecture:** The general RAAM architecture is defined for encoding and decoding trees of arbitrary branching factors. Here k branches, each of n units, are compressed into n units.

output layers which both contain k groups of units—one group for each possible branch of a tree node—with n units in each group, for a total of $k \times n$ units in each layer. The hidden layer consists of a single group of n units (see Figure 3).

Starting with the representations for the leaves, we recursively construct representations for each intermediate branching node by compressing the representations for each of its k child nodes into a single representation for the branching node, of length n. The encodings created for the internal branching nodes are saved and placed in the appropriate input slots when creating the encoding for their internal parent node. We continue this process up the tree until we have a compressed representation for the root node, which corresponds to a single, encoded representation for the entire tree structure. We can encode a number of trees using the same RAAM by simply performing this process for every tree we would like to encode.

Once we have an encoded representation of a tree, we may reconstruct the entire tree by utilizing the trained connections from the hidden layer to the output layer in the RAAM, which effectively serve as a decoding mechanism. The representation for the root node is first placed on the hidden layer, and the representations for each of the root node's children are recovered by propagating the root node pattern through the hidden/output layer connections. The rest of the tree is then recovered by recursively performing this operation on each of the resulting child node representations, until the leaves are recovered. A decision procedure must be specified to determine when leaves have been recovered. Since the representations for the leaves were chosen a priori, the decision procedure merely specifies the allowable range of continuous numerical values which will designate one of the original representations. Concrete examples of the encoding and decoding procedures will be presented in the next section.

The encodings produced by a RAAM do not explicitly reflect the structure they represent. Understanding the implicit structure in the representations often requires the use of analytical techniques such as cluster analysis and principal component analysis. In the next section we describe a very simple experiment in which the RAAM encodings are small enough to be dissected and examined without resorting to these more complicated analytical tools.

3 Analysis of a Simple Sequential RAAM Model

In the experiments described in this chapter, we shall use a slightly restricted version of the general RAAM model discussed above, called a *sequential RAAM*. The sequential model is just like the general version, except that the data structures which are stored and retrieved are ordered lists or *sequences* of elements, rather than general trees. Any sequence of elements can be represented as a simple left-branching or right-branching binary tree, so the sequential model is really just a special case of the more general model. However, in the general model, the size of the hidden layer representations must be exactly the same as that of the representations chosen for the leaf elements, because for arbitrary trees it is impossible to know beforehand whether a particular branch out of a branching node will contain the representation for a leaf, or the representation for some other branching node. This is not the case for a binary tree representation of a sequence of elements, since the left (or right) branch always contains the representation for some leaf, and the other branch always contains the representation for some other branching node. Thus, the representations chosen for our sequence elements, and the compressed representations created by the network for the sequences themselves, can differ in length. One advantage of employing the sequential model is that it allows us to use a larger number of processing units for the compressed hidden representations, in order to improve the information storage and retrieval capacity without having to change the representational scheme chosen for the sequence elements. There is also no need to arbitrarily decide how the sub-trees will be grouped; the sequential model will always branch to the side.

It is useful to consider how a sequential RAAM is analogous to a stack[4] data structure. The compression step is like the stack *push* operation and the decoding step is like the stack *pop* operation. When a sequence of elements (represented as a left or right branching tree) is compressed, the next element to be pushed and the current stack are given to the RAAM as input. It then creates an updated version of the stack, containing the new element, on its hidden layer. When a sequence is decoded, the representation of a stack is placed on the hidden layer. The topmost element and the remainder of the stack are produced as output.

To gain some intuition into the types of representations formed by a sequential RAAM, we devised the following very simple experiment. We trained a RAAM to encode sequences of two symbols, A and B. The symbols were represented by a single *bit*, a binary digit, either 0 or 1. The training sequences consisted of all the possible combinations of length three of the two symbols (see Table 2.) The input and output layers contained three units each—one to represent the symbol (either A or B) and two to represent the encoded internal tree nodes. The hidden layer contained two units. This architecture is referred to as a *3-2-3 RAAM* and is pictured in Figure 4.

After many learning trials with the training corpus, the RAAM's representational accuracy was tested by first encoding one of the sequences into a compressed representation

[4]A stack is a data structure similar to a plate dispenser at a cafeteria. You may push a plate onto the top of the stack, or you may remove (pop) a plate from the top. This is also known as last-in-first-out processing.

Number	Sequence
1	AAA
2	BAA
3	ABB
4	BBB
5	AAB
6	BBA
7	ABA
8	BAB

Table 2: Sequences in the Training Corpus of the 3-2-3 RAAM

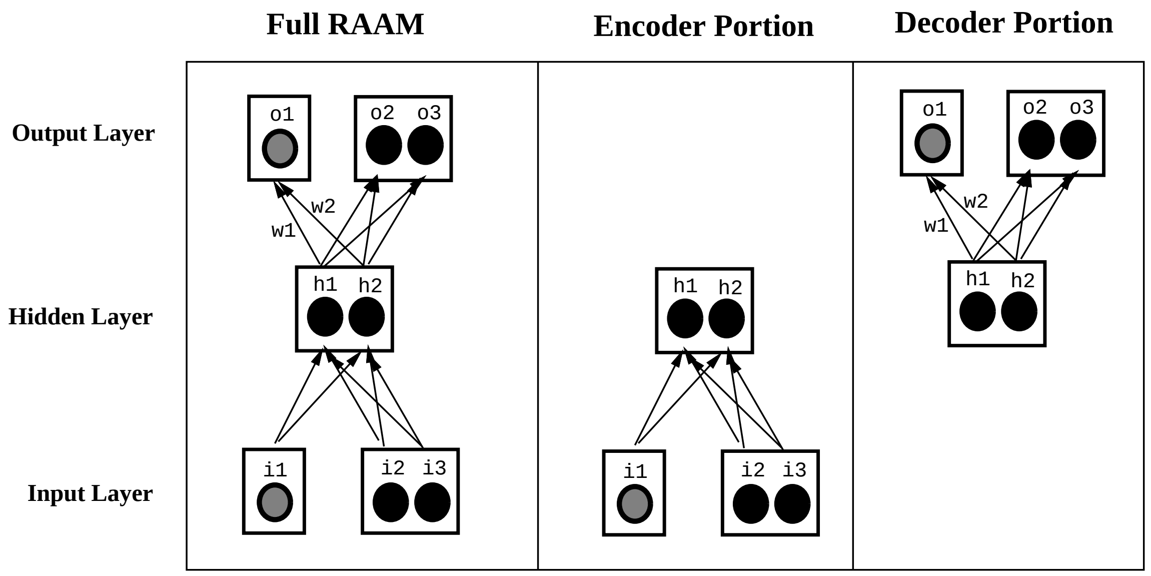

Figure 4: **3-2-3 sequential RAAM:** This simple sequential RAAM takes in one symbol at a time on the *i1* unit. Units *i2* and *i3* take in the previous hidden layer activations (from units *h1* and *h2*). The leftmost diagram shows the entire RAAM network; the middle diagram shows the encoder portion of the network; and the rightmost diagram shows the decoder portion of the network.

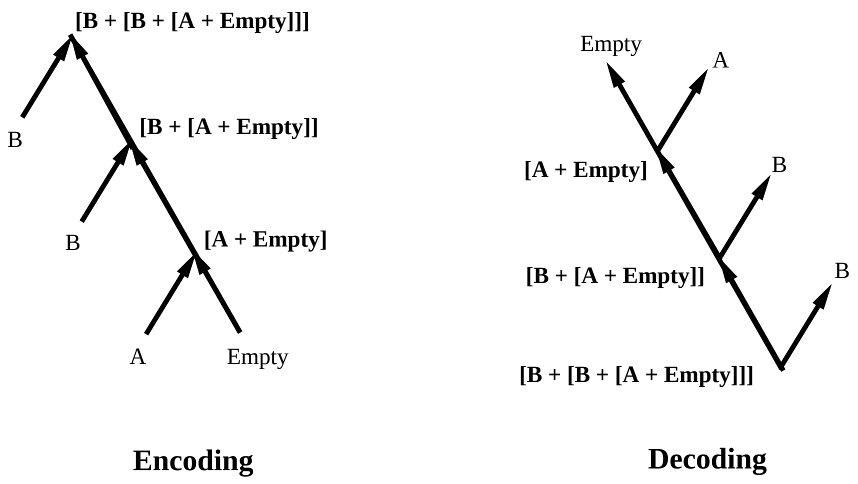

Figure 5: **Encoding and decoding in 3-2-3 RAAM:** The left half of this figure depicts the encoding of the sequence *ABB*, and the right half depicts the decoding of the same sequence. Notice that the decoding removes the symbols in the reverse order from that of the encoding.

and then immediately decoding that representation back into its constituent elements. If the decoded constituents match the original sequence elements, then the RAAM has learned to adequately represent that particular sequence.

Figure 5 depicts the steps required to encode and decode one of the sequences, *ABB*, in the 3-2-3 RAAM. To encode it, the representation for an *A*, a 0, was placed in the input symbol slot (unit *i1* in Figure 4), and the representation for the empty stack, (0.25, 0.25), was placed in the previous hidden layer slot (units *i2* and *i3* in Figure 4). Propagating these activations forward produced a compressed representation of *[A+Empty]* on the hidden layer. Next, the representation for *B*, a 1, was placed on unit *i1*, and the representation of *[A+Empty]* created in the previous step was placed on units *i2* and *i3*. Propagating these activations forward produced a compressed representation of *[B+[A+Empty]]* on the hidden layer. Finally in the third encoding step, a compressed representation of the entire sequence is obtained. To decode, this compressed representation *[B+[B+[A+Empty]* is placed on the hidden layer and activations are propagated forward to produce activations on the output layer. Decoding the entire sequence also requires three steps, as shown on the right of Figure 5.

The testing revealed that the RAAM was unable to learn all eight sequences perfectly. It produced decoding errors on the first element of sequences 4 and 8. This difficulty in mastering the training corpus was due to the very restricted size of the hidden layer. However, because the hidden layer was limited to two units, we could examine the RAAM's encodings directly by plotting them in a two-dimensional graph, as follows.

First, a graph was constructed in which the y-axis represented the value of hidden unit *h2* and the x-axis represented the value of the hidden unit *h1*.[5] Both of these values ranged

[5]This area is often called the *representational space* of a connectionist network. For every unit in the hidden layer, there is a dimension in representational space. This example, containing two hidden units, can be mapped in two dimensions.

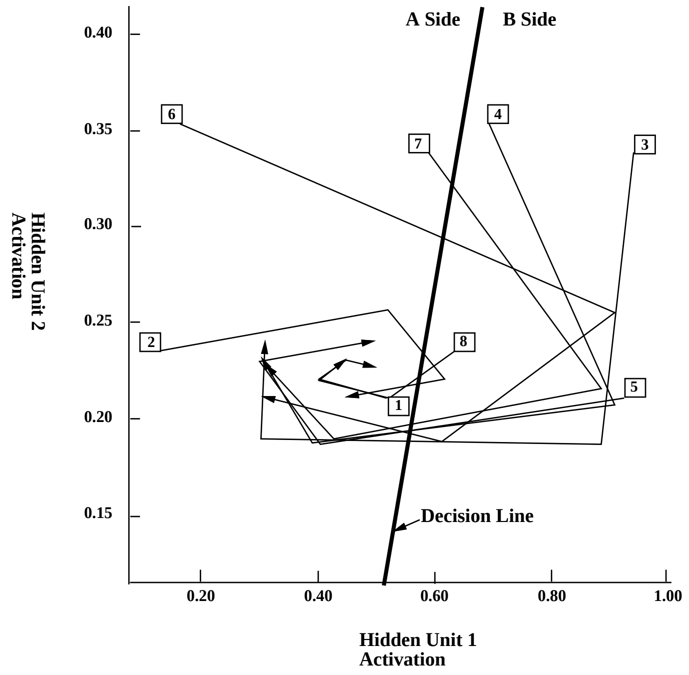

Figure 6: **Plot of all 8 sequences of the 3-2-3 RAAM:** This diagram shows the decoding plots of each of the sequences from Table 2. Each plot begins with its sequence number in a small box, travels in a clockwise motion, and ends at an arrowhead. The decision line defines where the activation of unit *o1* (see Figure 4) is 0.5.

between 0.0 and 1.0. Next, the RAAM's encoding of an entire sequence was plotted as the first point. Then the RAAM was allowed to decode the last item in a sequence.[6] After decoding the last item, the encoding of the remainder of the sequence appeared on the two rightmost output units (*o2* and *o3* in Figure 4). This was plotted as the second point. Then the middle item of the sequence was decoded. Again the encoding of the remaining sequence was plotted. This process was continued until the entire sequence had been decoded. Thus each plot contains 4 points: (1) the encoding of the tree which represents the entire sequence; (2) the encoding of the tree which represents the first two elements; (3) the encoding of the tree which represents the first element; and (4) the encoding of the empty tree.

Figure 6 shows the decoding plots formed by all eight sequences trained in the 3-2-3 RAAM. The numbered boxes mark the beginning and the the arrowheads indicate the end of each plot. Notice that each plot moves in a clockwise progression through the space. Let us examine sequence 3, *ABB*, more closely. The point at the box represents the entire sequence, in reverse order as on a stack—*BBA*. The next point represents the encoding of the remainder of the tree after the final element has been decoded or popped off the stack—*BA*. The third point represents the tree after the middle element has been decoded—*A*. The final point represents the empty tree.

How has the RAAM partitioned the hidden layer space to represent these sequences?

[6]Recall that the RAAM builds up its representation of a sequence from left to right, but when it decodes the sequence it must reverse the process. So the elements of the sequence are returned in reverse order.

One way to determine this is to examine the activations of unit *o1* in the output layer of the RAAM. When the activation of *o1* is less than 0.5, an A has been decoded (since an A is designated by a 0), and when the activation is greater than 0.5 a B has been decoded (since a B is designated by a 1). If we could determine when unit *o1*'s activation is 0.5, we could see how the RAAM has used the hidden layer activations to encode the sequences. Referring to Figure 4, *o1*'s activation is given by the equation:

$$act(o1) = sigmoid((act(h1) \times w1) + (act(h2) \times w2) + bias(o1))$$

where $act(x)$ is the activation of unit x, $bias(x)$ is the internal threshold of unit x, and $sigmoid(y) = \frac{1}{1+e^{-y}}$. By setting $act(o1)$ to 0.5 and substituting in the known values for $w1$, $w2$, and $bias(o1)$, the following equation is found, representing a line in the space of hidden unit activations:

$$act(h2) = (1.78 \times act(h1)) - 0.801$$

This line has been plotted in Figure 6 and labeled as the "Decision Line," since it divides the representational space into an "A Side" and a "B Side."

So we can now see that the RAAM has clustered all of the encodings according to whether the topmost element in the stack is an A or a B. All points to the right of the diagonal line designate encodings containing a B as the next element to be decoded, while all points to the left of the diagonal designate encodings containing an A as the next element to be decoded. The two sequences that were decoded incorrectly, 4:*BBB* and 8:*BAB*, both returned an A as the first element rather than a B. This is reflected in the fact that the third point in each of their plots is to the left of the diagonal, instead of to the right, as it should be.

The RAAM has also partitioned the hidden layer space in at least one other way: all plots which begin above the horizontal line $act(h2) = 0.30$ (this line is not shown in the figure), represent sequences whose middle element is a B (sequences 3, 4, 6, and 7). Similarly, all plots which begin below this horizontal line represent sequences whose middle element is an A (sequences 1, 2, 5, and 8).

Even in this very simple experiment, the richness of subsymbolic representations is evident. Hidden unit *h2* can be viewed as a subsymbol which encodes the middle element in the sequence. The combination of both hidden units can be viewed as a symbol encoding the current top-most element in the sequence. Additionally, the two hidden units can be seen as a symbol for the entire composed sequence structure. All of this information is simultaneously represented in the continuous numerical values of two simple units with respect to the RAAM's weighted connections.

It is interesting to note that the values we chose to represent the empty tree had a dramatic effect on how the RAAM partitioned the hidden unit space, and on its ability to accurately represent all eight sequences. Recall that the empty tree values are the initial values placed on units *i2* and *i3* in the first step of encoding a sequence. For the results discussed above, the values 0.25 and 0.25 were used to represent the empty tree. Figure 6 shows that all eight sequence plots do, in fact, end in the vicinity of this point. We ran other experiments in which both of the empty tree values were 0.0, 0.5, 0.75, and 1.0. In the 0.0 case, the RAAM could learn only four of the sequences correctly, and the decision line was much closer to horizontal (rather than vertical as in the 0.25 case). When we tried the same experiments with a 4-3-4 RAAM, these effects were not found, suggesting that if given enough hidden units for a given task, a RAAM can adequately represent the sequences without regard to the particular empty tree values chosen.

4 Generalization

One of the most useful qualities of connectionist models such as RAAM is their ability to learn a set of training examples and then to generalize from this training corpus to produce appropriate output for novel inputs. Connectionist models are often evaluated on their generalization performance. If a model can respond suitably to the majority of novel inputs (typically a performance of 75% or greater is considered good), then it must have developed a veridical representation of the entire task environment and not simply of the trained portion. More importantly, even when a connectionist model makes mistakes on novel inputs, these mistakes often reflect generalizations of the training task (as we will see in Section 5.3).

Several factors affect a network's ability to produce useful generalizations: the complexity of the environment to be modeled, the contents and size of the training corpus, the number of exposures to the training corpus, and the size of the hidden layer (in three-layer feed-forward networks). If the training corpus does not contain a diverse enough sampling of the possible inputs, the network will only be able to respond reasonably well to novel inputs which are closely related to the training examples. If the network is trained until it responds perfectly to all of the training examples, or if the hidden layer is very large, it may succeed at the training task by simply memorizing every example, rather than forming useful generalizations. Finally, as we have already seen with the 3-2-3 RAAM, if the hidden layer size is too restricted, the network may not be able to learn the training examples adequately, although it may still form useful generalizations about the examples it does learn.

Consider the 3-2-3 RAAM once more. Since it was trained on all of the possible sequences of length three, there were no remaining novel sequences with which to test its generalization abilities. Instead, we devised a different kind of generalization test. We again represented the hidden layer activation space as a two-dimensional graph, just as in Figure 6. Then we selected a sample point from this activation space, placed it on the hidden layer of the trained RAAM, and decoded it for a single step. This produced another point in the hidden layer activation space on the output units *o2* and *o3*. We plotted the original point and its decoded result as a vector with the arrowhead ending at the decoded result. Figure 7 shows how a large number of these sample points decoded to produce the next point in sequence. Although most of these sample points did not correspond to any of the trained sequences, the RAAM's outputs all followed the same trends observed for the trained sequences. The vector field moves in a counterclockwise direction through the space; vectors originating near the edges of the space are typically long, often crossing the decision line; and the overall tendency of the field is to converge towards the vicinity of the empty tree point (0.25, 0.25). The 3-2-3 RAAM's responses to novel inputs closely follow its responses to trained inputs. It has formed generalizations of the training data in a way that allows it to make reasonable responses to novel data.

In the remainder of the experiments to be described, we use generalization tests as one method for evaluating: (1) whether the RAAM itself has developed useful representations, and (2) whether additional networks which take these RAAM representations as input can

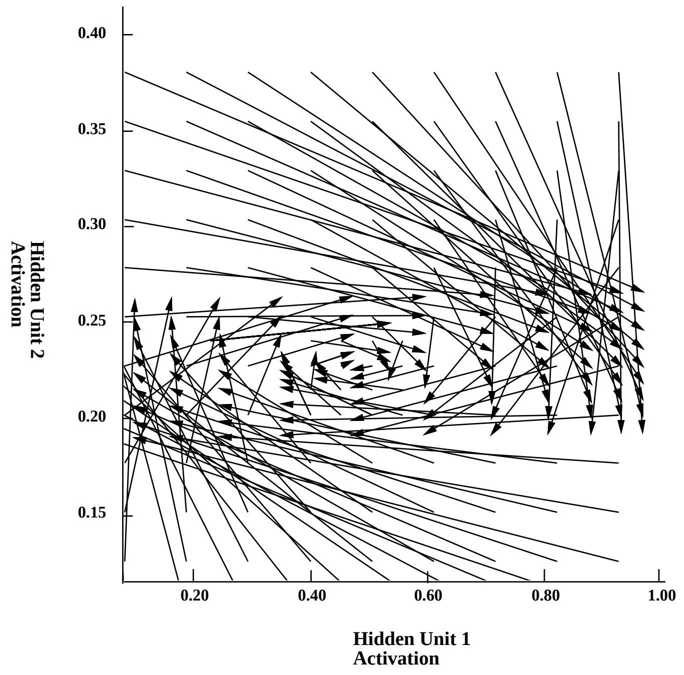

Figure 7: **Generalization in the 3-2-3 RAAM:** The same technique as shown in Figure 6 was used, except that the decoding was begun at points all over the plane and was followed for only one step. This illustrates the generalization ability of the network.

generalize over them in an interesting way.

5 RAAM-encoded simple sentences

The simple sequence experiments performed on the 3-2-3 RAAM reveal some of the potential offered by subsymbolic representations. To explore this potential more fully we devised a similar but more complicated set of experiments where sentences were represented as sequences of words. A corpus of two and three word sentences was created using a small grammar, a *tarzan grammar* if you will, and then encoded by a sequential RAAM. Here we will describe the generation of the sentences from the grammar, specify the RAAM architecture used, and perform an analysis of the RAAM's encodings of the sentences.

We chose natural language as the domain for these experiments because it requires complex, structure-sensitive operations that have traditionally been modeled in the symbolic paradigm. These RAAM experiments are presented to illustrate the contrasting styles of representation, composition, and functionality available in subsymbolic models, and should not be construed as a linguistic model.

5.1 Creation of Simple Sentences

A trivial program to generate English-like sentences was written to produce two and three word sentences from a set of 26 words (15 nouns and 11 verbs). An additional word, *stop*, was used to indicate the end of a sentence. The words were described by the lexical categories

Category	**Members**
NOUN-ANIMATE	*tarzan jane boy cheetah chimp rhino*
NOUN-AGGRESSIVE	*cheetah rhino bigfoot junglebeast*
NOUN-EDIBLE	*coconut banana berries meat*
NOUN-SQUISH	*banana berries*
NOUN-MOBILE	NOUN-ANIMATE + *(bigfoot junglebeast jeep)*
NOUN-SWINGER	*tarzan chimp*
NOUN-HUNTER	*jane*
NOUN	NOUN-ANIMATE + *(bigfoot junglebeast)* + NOUN-EDIBLE + *(jeep tree rock)*
NOUN-REAL	NOUN − *(bigfoot junglebeast)*
VERB-FLEE	*flee*
VERB-HUNT	*hunt*
VERB-AGGRESS	*kill chase*
VERB-SQUISH	*squish*
VERB-MOVE	*move*
VERB-EAT	*eat*
VERB-PERCEIVE	*see smell*
VERB-INTRANS	*see smell*
VERB-EXIST	*exist*
VERB-SWING	*swing*

Table 3: Categories of lexical items used in sentence generator.

Template	**Word1**	**Word2**	**Word3**
1	NOUN-ANIMATE	VERB-FLEE	NOUN-AGGRESSIVE
2	NOUN-AGGRESSIVE	VERB-AGGRESS	NOUN-ANIMATE
3	NOUN-ANIMATE	VERB-SQUISH	NOUN-SQUISH
4	NOUN-ANIMATE	VERB-EAT	NOUN-EDIBLE
5	NOUN-ANIMATE	VERB-PERCEIVE	NOUN
6	NOUN-MOBILE	VERB-MOVE	
7	NOUN-ANIMATE	VERB-INTRANS	
8	NOUN-REAL	VERB-EXIST	
9	NOUN-SWINGER	VERB-SWING	
10	NOUN-HUNTER	VERB-HUNT	
11	NOUN-AGGRESSIVE	VERB-HUNT	

Table 4: Templates used in sentence generator.

given in Table 3. Note that one word may belong to several different categories. For instance, *cheetah* is a member of the following categories: NOUN-ANIMATE, NOUN-AGGRESSIVE, NOUN-MOBILE, NOUN-REAL, and NOUN. The sentence templates given in Table 4 specified the ways that the lexical categories could be combined to form sentences. All of the sentences were either of the form (NOUN VERB) or (NOUN VERB NOUN). There are 26^2 possible two-word sequences and 26^3 possible three-word sequences for a total of $18,252$ possible sequences; the grammar restricted this to a very small subset of 341 valid sentence. This grammar was inspired by one used by Elman (1990) in his experiments on sentences for his simple recurrent network architecture (see also Lee and Gasser, this volume). Table 5 gives some examples of typical sentences produced by the generation program.

To represent these words as input to a RAAM, each word was randomly assigned an individual code. The code consisted of 27 bits. We used a *localist* representation; for each word one bit was on, the other 26 were off. Each input symbol was orthogonal to all of the others and had no microsemantics. For the majority of the experiments described below, the corpus consisted of 100 unique sentences to be used for training and another 100 to be used for generalization tests.

5.2 RAAM Processing of Sentences

The structure of the RAAM used is shown in Figure 8. The input is divided into a set of 27 word units and a set of 30 encoded units (from the previous hidden layer activations). Since the words were given a localist representation, only one of the word units was on at a time. The number of words used determined the number of word units needed. Determining an appropriate number of encoding units was not as straightforward; the number should be large enough to allow the RAAM ample space to successfully compress and reconstruct trees of the required depth, but small enough to allow useful generalizations to develop. The number of encoding units was chosen by sampling the RAAM's performance over a small number of tests with the hidden layer size ranging from 20 to 50. By this trial-and-error process, the hidden layer size was set to 30 units.

Recall that this is not a general RAAM, but a sequential RAAM (the left set of units

Template	Example sentence
1	jane flee junglebeast
2	cheetah kill chimp
3	boy squish banana
4	rhino eat meat
5	bigfoot see jeep
6	tarzan move
7	chimp smell
8	tree exist
9	tarzan swing
10	jane hunt
11	junglebeast hunt

Table 5: Sentences generated from the categories and templates.

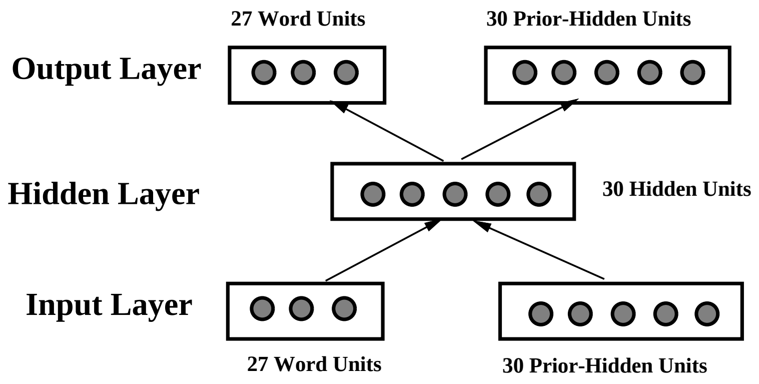

Figure 8: **RAAM architecture used for simple sentence experiments:** Sentences from the *tarzan grammar* were encoded using this network architecture.

always encodes a single element and the right set of units encodes the rest of the binary tree). The primary reason we chose to use the sequential model rather than the general model was that we didn't want to make any prior commitment to a particular syntactic structuring. With the general model, it would be necessary to structure the input as some form of parse tree, with the internal units representing syntactic groupings such as noun phrases or verb phrases. We wanted to make as few assumptions as possible about the form of the input, and so the simpler sequential version was more appropriate.

Treating each sentence as a sequence of words, we trained a 3-2-3 RAAM network to read in one word at a time, from left to right, as in the simple RAAM experiment above. The 100 training sentences were presented in a random order. The hidden units were cleared between sentences. After approximately 21,000 presentations of each sentence, training was halted.

To test the accuracy of the RAAM's performance, the sentences were encoded into a distributed representation and decoded back into their localist representation. A decoding was considered an error when the activation of the correct word unit did not exceed 0.5, or if another word was more strongly active than the correct word unit. The 100 trained sentences and 80% of the 100 new sentences were decoded perfectly. In 15 of the new sentences which caused errors, all of the correct word units were activated, but the activation was not above 0.5 for at least one of the words in each sentence. These errors could probably be alleviated if the RAAM were to be trained for a longer time. In each of the other 5 sentences which produced decoding errors, one word was decoded as another word. This is a much more serious error. Some of these severe errors can be explained by examining the training corpus more closely. The average frequency of the 11 verbs in the 100 trained sentences was 9%. The average frequency of the 15 nouns was 12.3%. In two of the five errors, the RAAM decoded *jane* rather than the correct word *jeep*. In another, it decoded *flee* instead of *swing*. The frequency of *jeep* (1%) was well below the average for nouns. The RAAM may not have had enough examples over which to generalize a representation of *jeep*. In the *swing* case, the random sentence generator did not generate a single sentence containing that particular word for the training corpus. Therefore it is not surprising that the RAAM was unable to decode it correctly when it was presented with it for the very first time.

It is interesting to note that even when the RAAM made a serious mistake such as decoding the incorrect word, the word it usually returned was of the same grammatical type. For instance, a noun was often substituted for another noun (i.e. *jane* for *jeep*.) Thus it appears that the RAAM is making useful generalizations in its encodings.

One final set of generalization tests reveals the context-sensitive quality of the RAAM's compositions. We created 20 ungrammatical sentences and tested whether they could be decoded correctly. Some of these sentences were ungrammatical in very subtle ways. For example *tarzan chase bigfoot* is ungrammatical because *tarzan* is not a member of the NOUN-AGGRESSIVE category, and *bigfoot exist* is ungrammatical because *bigfoot* is not a member of the NOUN-REAL category. The majority of the test sentences were ungrammatical in more obvious ways. For instance *berries chase meat* is ungrammatical because *chase* requires a subject from the NOUN-AGGRESSIVE category and an object from the NOUN-ANIMATE category, while *eat tree eat* does not even follow the (NOUN VERB NOUN) sentence structure. The RAAM could only decode 35% of the novel ungrammatical sentences correctly as opposed to the 80% reported earlier for the novel grammatical sentences. Of the ungrammatical sentences decoded correctly, 86% were only subtly ungrammatical. Since the RAAM had difficulty encoding and decoding ungrammatical sentences, it appears that it has taken advantage of the regularities in the grammatical training corpus to form its hidden representations. Clearly in these experiments, the RAAM's ability to accurately represent novel sentences was context-sensitive. However, RAAMs can exhibit systematicity within the training context and on novel inputs which mirror the training environment [Chalmers, 1990a].

5.3 Analysis of Sentence Encodings

To examine the RAAM's development of generalizations more directly, a cluster analysis, based on Euclidean distance, was performed on the encoded representations of the 100 trained sentences. This procedure clusters similar patterns together for comparison, collapsing a 30-dimensional space into a compact tree structure. Figure 9 shows the general structure of the entire clustering, and one sub-cluster of 26 sentences in detail.

At first glance, the clustering seems to be based mostly on the last word of the sentence. For example, there is a large cluster of sentences near the bottom of Figure 9, all of which end in *boy*. Although this is sometimes the case, it is not the whole story. For instance, the middle cluster, which includes *rhino see rhino*, contains sentences of the form: aggressive nouns perceiving aggressive nouns. Some of these sentences end with *rhino* and some with *bigfoot*. Two other sentences ending with *rhino* (*bigfoot chase rhino* and *bigfoot kill rhino*) have been clustered separately. These two separate sentences contain aggressive verbs, rather than perceiving verbs as in the middle cluster. Sentences ending with the word *meat* have also not all been clustered together; the smelling of meat has been separated from the eating of it.

On the portion of the cluster analysis not shown in detail, all sentences ending with the words *berries* and *banana* have been clustered together. This cluster is quite far away from the one shown on the right side of the figure. The other two edible nouns were *coconut* and *meat*. The only difference in the ways these two sets of nouns were used was that *berries* and *banana* were squishable while *coconut* and *meat* were not. Clearly the RAAM's representations of these words are sensitive to the context in which they are used.

Another indication of the RAAM's attention to context is evident in its representations of sentences of the form NOUN-ANIMATE *eat meat* and NOUN-ANIMATE *smell coconut*. In both of these clusters, the sentences in which NOUN-ANIMATE is also aggressive are closer together than those sentences in which NOUN-ANIMATE is non-aggressive. So in sentences where the last two words are identical, the RAAM is using both the actual first word and its context to disambiguate the sentences.

In summary, the cluster analysis of the sentence encodings provides some evidence that the RAAM has developed generalizations of aggressive animates, squishable edibles, and aggressive verbs. However, using the clustering of the sentence representations to make inferences about the word representations is somewhat indirect. Elman has designed a more direct method to examine his simple recurrent network's internal representations of words [Elman, 1990]. We also applied this method to the RAAM.

For this method, the 100 sentences in the training corpus were again passed through the encoder portion of the RAAM after the completion of training. The hidden layer activations created by each word combined with the context of the current sentence were saved. For the 286 words in the 100 sentences, we obtained 286 numerical vectors of length 30 (the size

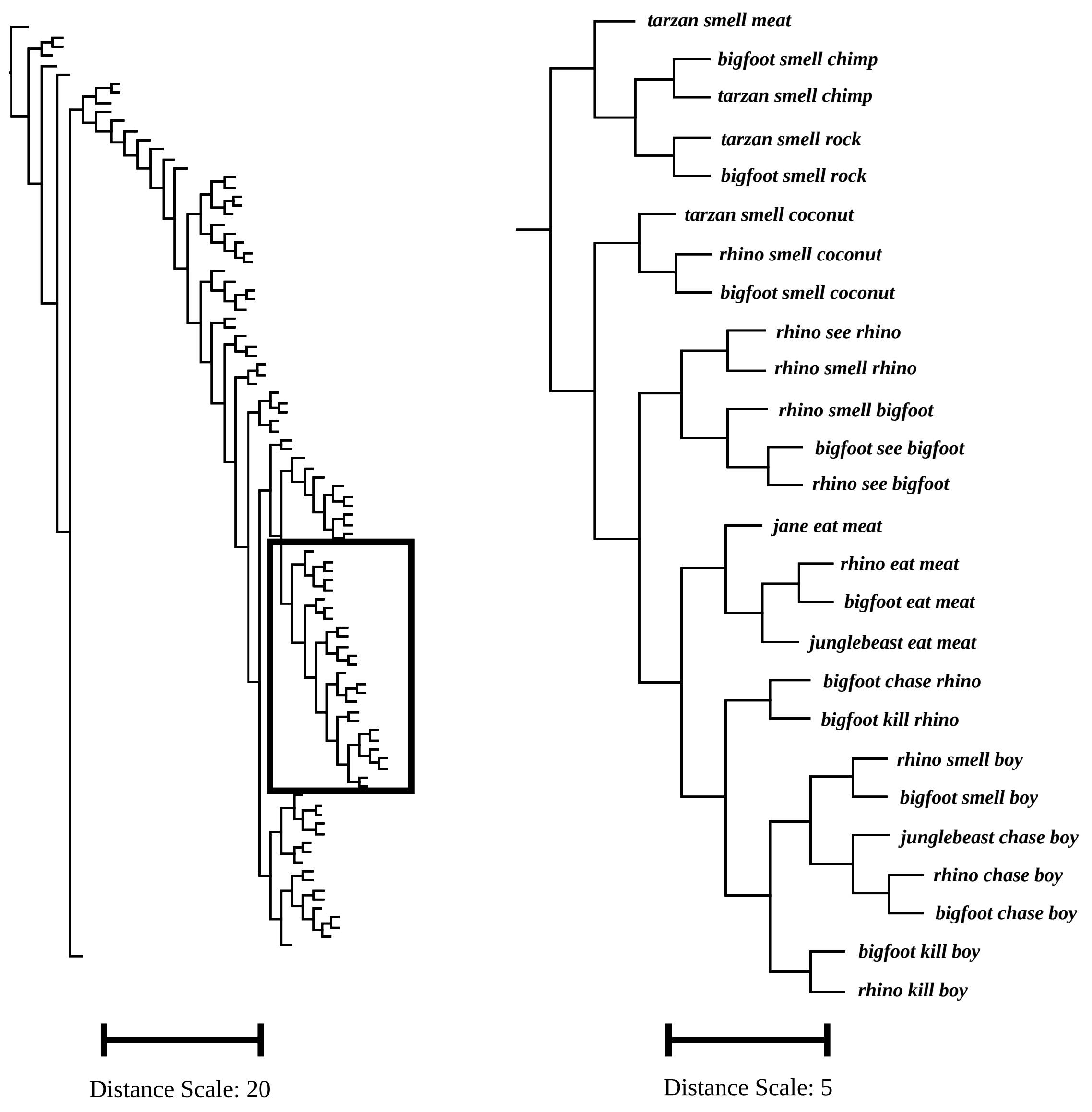

Figure 9: **Sentence cluster analysis:** A clustering of the RAAM's representations of the sentences of the grammar is shown. The left tree shows the entire cluster of the 100-sentence training corpus, while the right tree is an enlarged view of the boxed area.

of the hidden layer). All of the vectors produced by a particular word were then averaged together to create one composite vector for each of the 26 unique words. Each composite vector reflects all of the contexts in which a word has been used in the 100 training sentences. These composite vectors are shown schematically in Figure 10. Some similarities between these composite vectors stand out quite clearly; the activation of the sixth hidden unit (shown in the sixth column in Figure 10) seems to encode whether the word is a verb or a noun. For all of the verbs, the activation is quite high, and for all of the nouns (with the exception of *jeep*), the activation is quite low. As noted previously, *jeep* only appeared once in the training corpus, so the RAAM did not have an adequate number of examples to form an appropriate representation for this word.

To further examine the similarities between these composite word vectors, another cluster analysis was done (see Figure 11). Again it is clear that the RAAM has made a distinction between verbs and nouns, although there are eight exceptions. Some of these exceptions (*hunt*, *jeep*, and *tree*) appeared infrequently in the training corpus and can be explained in this way. However, the other exceptions appeared more frequently than the average for verbs and nouns. There are several possible explanations for these anomalies: (1) perhaps the small size of the training set combined with the relatively large size of the hidden layer allowed the RAAM to develop special representations for some of the words, and (2) perhaps the 100 sentences randomly selected for the training corpus were not representative of the grammar. In either case, increasing the size of the training corpus should eliminate or decrease the anomalous cases.

We tested this hypothesis by creating a new training corpus containing 300 randomly selected sentences (from the 341 possible sentences) and training a new RAAM. As was expected, the clustering of the composite word vectors for the RAAM trained on the larger corpus had fewer exceptions (five as opposed to eight). In addition, only two of these exceptions, *jane* and *cheetah*, were the same. The size and contents of the training corpus clearly affected the RAAM's ability to generalize. In Elman's experiments in which he used a similar grammar, the training corpus contained 10,000 sentences, and all of the network's composite representations of nouns and verbs appeared in separate clusters [Elman, 1990]. Unfortunately, the simple tarzan grammar we devised restricted the number of valid sentences drastically, so such an extensive test was not possible.

Further examination of Figure 11 reveals that the RAAM has made more than just a noun/verb distinction; within the noun cluster there are additional discriminations. The squishable foods, *banana* and *berries*, have been clustered together, as have the non-squishable foods *coconut* and *meat*. The aggressive nouns within the noun cluster, *bigfoot* and *rhino*, are also clustered together.

Although the RAAM was never given any explicit information about the tarzan grammar, the cluster analyses reveal that its representations of the sentences and words do reflect the grammar's rules to a certain extent. How the words were used (their context) in the example sentences directly affected the form of the RAAM's representations. Furthermore, the cluster analyses shows that the representations do possess a microsemantics.

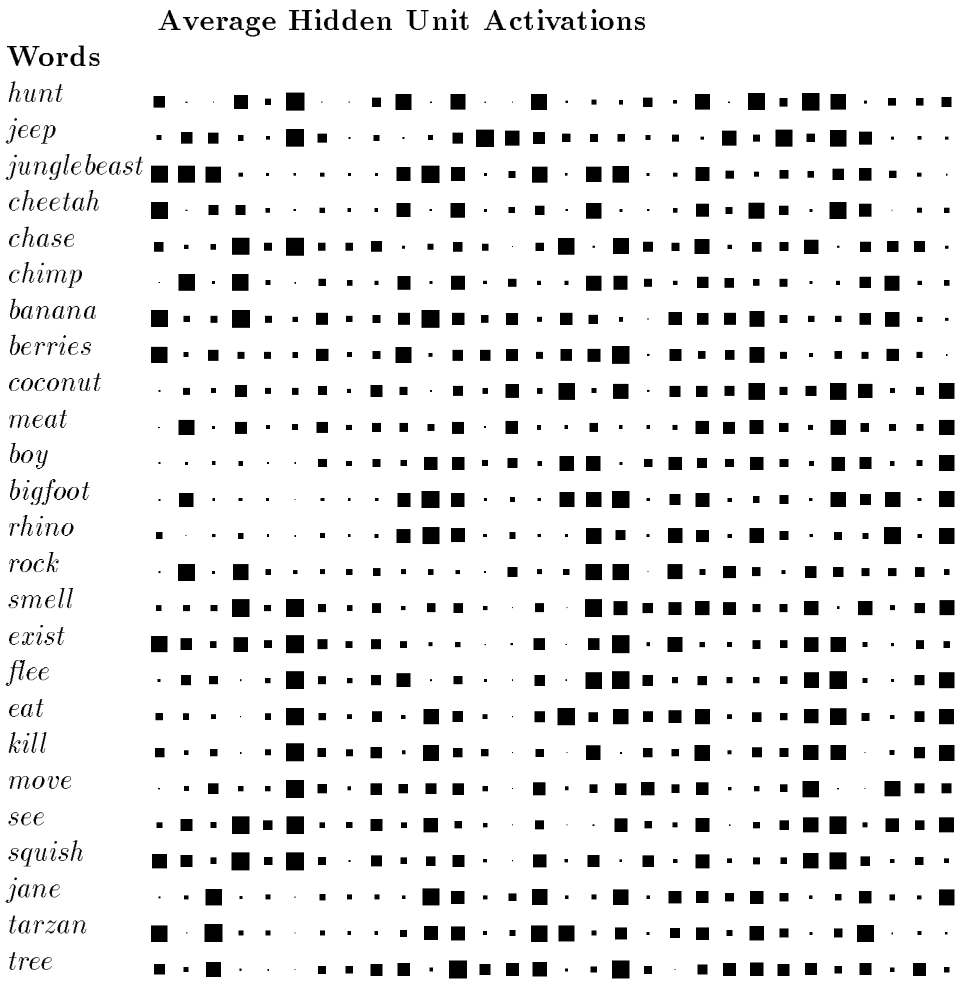

Figure 10: **Composite word vectors:** The average hidden layer activations associated with each word in its various contexts are depicted. One column of rectangles is shown for each of the 30 units in the hidden layer. All activations fall in the range from 0 to 1. The area of a rectangle encodes the magnitude of the average activation.

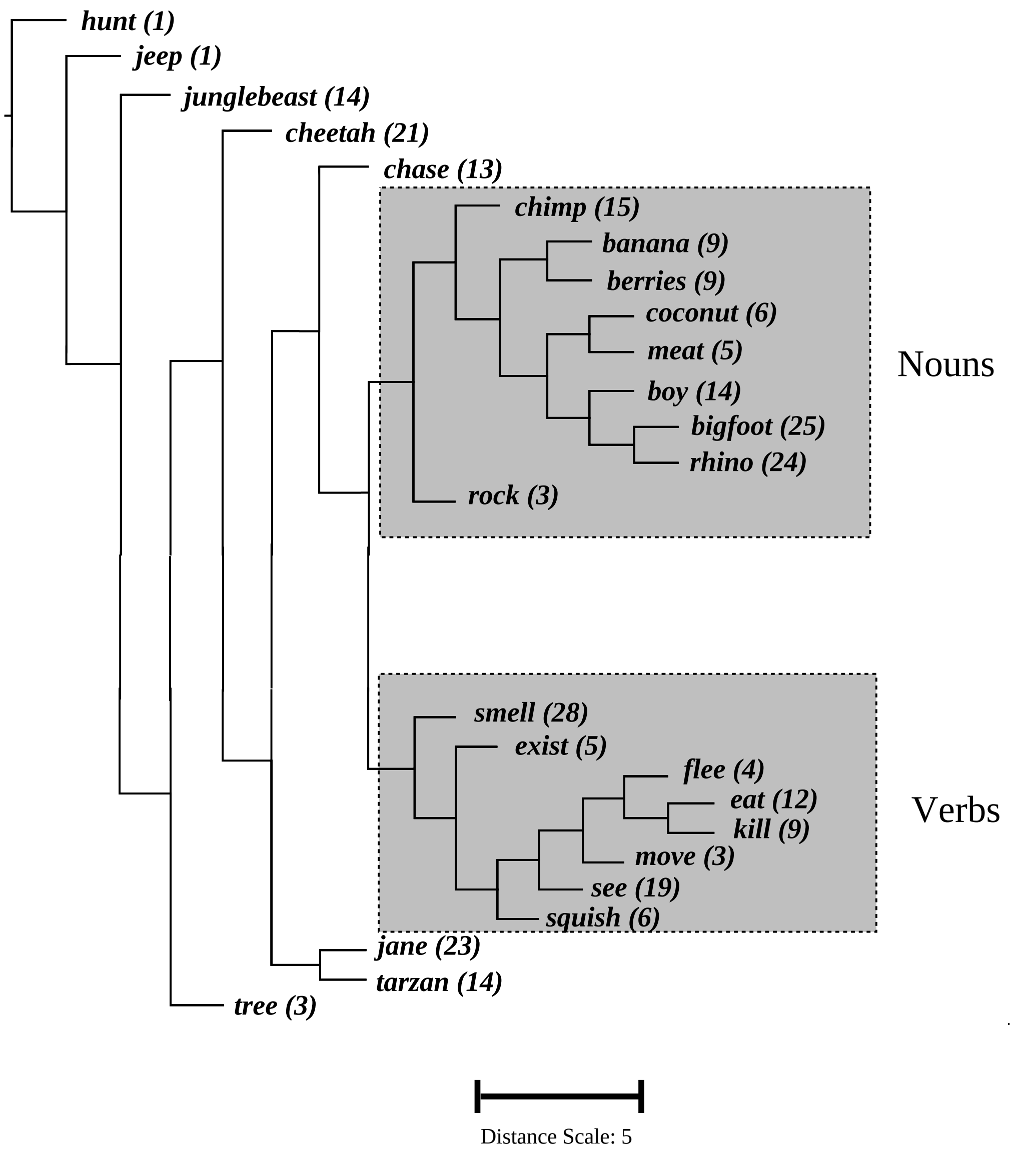

Figure 11: **Composite word vector cluster analysis:** The clustering of the average hidden layer activations associated with each word in its various contexts is shown. The number in parentheses following each word indicates the number of times it appeared in the 100-sentence training corpus. This is another method of visualizing the information in Figure 10.

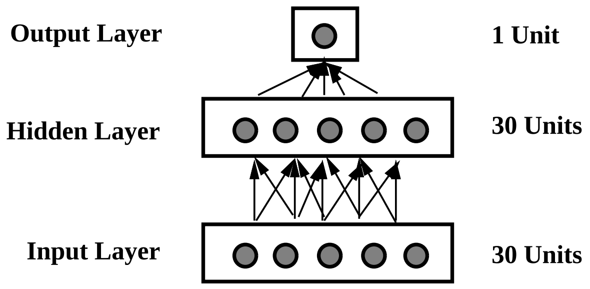

Figure 12: **Feed-forward network architecture for detectors:** The single output unit was trained to produce a 1 for positive instances of the detection tasks and a 0 for negative instances.

6 Holistic Operations

To explore the unique functional capabilities resulting from the accessible microsemantics in RAAM representations, we designed experiments which take RAAM encoded sentences as input and operate holistically on these representations to produce output. Three types of experiments will be described: detectors, decoders, and transformers. Each experiment was accomplished with a feed-forward network.

6.1 Detectors

All of the detection experiments followed the same basic outline: train a new feed-forward network to take an encoded sentence representation from the RAAM and produce a simple YES or NO as output if the presence or absence of some feature (or combination of features) is detected in the input. This was accomplished by having a single output unit that is trained to produce a 1 in a positive instance and a 0 in a negative instance. Figure 12 shows the basic architecture of each detector network.

6.1.1 Aggressive-animal Detector

The premise for this experiment is straightforward: given a sentence, is a member of the NOUN-AGGRESSIVE category present? For example, *tarzan eat banana* should produce a NO, while *junglebeast kill chimp* should produce a YES. Note that this is not a detector for an aggressive animal acting in an aggressive manner; the syntactic presence or absence of an aggressive noun is all that matters. So the sentence *bigfoot smell banana* is also a positive example for this detector.

The composed representation of a sentence was placed on the input units. Then the original sentence was examined to see if one of the aggressive nouns appeared in the sentence, and the output target was set accordingly. This feed-forward network was trained on 50 of the sentences from the training corpus for approximately 300 trials.

The other 50 sentences from the original training corpus were then tested in the network to see if the it had generalized the task. The network made 6 errors getting 88% of these novel sentences correct. Four of the errors were false NO's, and two were false YES's. No single characteristic seems to satisfactorily account for all of the errors.

Following this test of generalization, the 100 sentences not trained in the RAAM where encoded via the RAAM and tested in this network. This is a double test of generalization: firstly in the RAAM, and secondly in the feed-forward network. Surprisingly, the network performed just slightly less accurately than in the previous test, scoring 85% correct over the 100 sentences.

6.1.2 Aggressive-animal-and-human Detector

This detector is similar to the previous one with an additional constraint: is there a member of the NOUN-AGGRESSIVE category and a HUMAN in the sentence? This could be viewed as the logical AND of two smaller detectors where both must be positive for the entire detector to be positive. However, the network will of course build its own representation of the problem. The HUMAN category includes the nouns: *tarzan*, *jane*, and *boy* and was not explicitly part of the grammar.

The training proceeded exactly as above with the added constraints mentioned. The network was trained for approximately 300 trials and performance was nearly identical. Again, the network was tested on 50 RAAM-trained composite representations this time missing 8, for 84% correct. The network performed nearly the same on the 100 generalized RAAM sentences missing 15 of 100 for 85% correct.

It is not surprising that 5 of the 6 representations that caused errors in the aggressive-animal detector were also missed by the aggressive-animal-and-human detector.

It is interesting to note that the RAAM had trouble decoding some of the sentences that the detectors processed correctly. In fact, of the 15 errors that the RAAM decoder made, only two of those were ones that a detector had problems with also.

6.1.3 Reflexive Detector

Although both of the previous detectors trained easily and generalized quite well, the following experiment proved to be a tougher problem. A feed-forward detector was trained to produce a YES if the subject and the object of the encoded sentence were the same. To ensure that the corpus had adequate examples of these types of sentences and to remove two-word sentences, a new data set was created. This corpus consisted of 28 reflexive sentences(i.e., *junglebeast smell junglebeast*), and 40 non-reflexive sentences. The following verbs were considered reflexive: *flee*, *chase*, *kill*, *smell*, and *see*. Only the animate nouns plus *bigfoot* and *junglebeast* could be used reflexively (as defined by the grammar). Finally, the architecture of the detector network was slightly modified. The hidden layer was decreased to 15 units, in an attempt to force the network to make useful generalizations.

A new version of the sentence encoding RAAM from Figure 8 was trained on the 68 sentence corpus. These encoded sentences were then used to train the new detector network for 600 trials. After this training, two types of generalizations were tested: (1) generalizations over the verb, and (2) generalizations over the noun (or the reflexive word).

In the first case, the network was trained on the representation for sentences such as *bigfoot see bigfoot*, and then given the novel representation for *bigfoot smell bigfoot*. All of these types of encoded sentences were correctly detected as being reflexive. The network also correctly detected non-reflexive encoded sentences over a verb, and therefore generalized 100% correctly.

In the second case, the network was unable to master generalization over the noun. In the training corpus, *tarzan* and *jane* never appeared in the same sentence together, nor did either appear in the same sentence twice. The network was then given encoded representations for sentences of the form *tarzan X tarzan* and *jane X jane* where *X* was one of the possible reflexive verbs. The network did not respond positively to any of these generalization tests. When the network was given *jane X tarzan* and vice versa, it correctly responded with a negative. However, it responded negatively in all cases indicating that it was not performing the task satisfactorily.

After examining the network and the method of encoding, it seems that the network was not easily finding a strong relationship between the representation of a word used in the first position and the representation of the same word used in the third position. If a general relationship existed between these representations a straightforward method of deciding if the words were the same would exist. However, the network failed to make this generalization. We believe that this might have been caused by the fact that the grammar was too constrained. By developing a more complex grammar, and allowing for deeper trees, the network would have to conserve the representational space of each symbol. This might force it to develop generalized representations so that representations of the same word used in two positions would be more similar.

The three detector experiments show that a feed-forward network can detect the presence and absence of particular features in the composed sentence representations produced by a RAAM without decomposing the sentence into its constituent words. The question remains: Is it only generalized features of the composed structures that are directly accessible in subsymbolic representations, or are the actual constituents themselves accessible as well?

6.2 Parallel Decoding

We have seen that when a RAAM is trained, an encoder and a decoder are naturally created as a by-product of auto-associating through a compressed hidden layer. By using these trained connections, we are able to treat the hidden representation developed by a sequential RAAM as a *stack*, so that the last symbol encoded into the stack is the first one decoded. But is this sequential peeling-off of one symbol at a time the only method to retrieve information from the composite representation?

To address this question, we designed a feed-forward network to decode all of the words in composed sentence representation in parallel (see Figure 13). The architecture of this network is very similar to the detectors, however instead of having a single unit on the output layer, each word of the sentence is represented on the output layer from left to right. The first word of the sentence is trained to appear in the far left of the output, followed by the second word, and third (if one exists).

The encoded representations of the first 50 sentences from the original RAAM corpus were trained for approximately 7,200 trials. Attempting to decode the remaining 50 sentences could result in a possible 150 mistakes: for each of the three-word sentences the decoder had to correctly produce all three words and for each of the two-word sentences the decoder had to correctly produce the two words and fill the third output slot with no activation. The network successfully decoded all but 29 of the words for an overall score of 81% correct. When the network made a mistake, it again consistently selected a noun for a noun and a

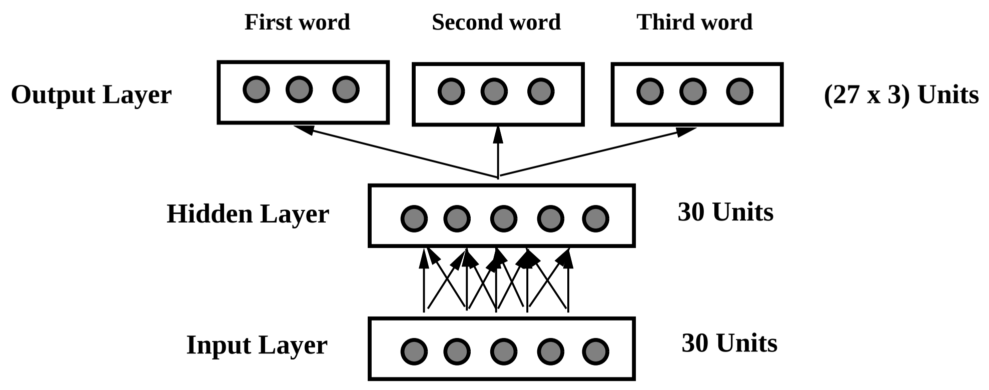

Figure 13: **Feed-forward network architecture for parallel decoding:** This shows the architecture for decoding composed sentences in one step. The words of each sentence were trained to appear on the Output Layer, from left to right. If only two words were in a sentence then the rightmost output slot was trained to give no activation.

verb for a verb, much like the original RAAM.

The parallel decoder experiment demonstrates that the actual constituents of composed structures in the subsymbolic paradigm are readily accessible.

6.3 Syntactic Transformations

So far the experiments employing the RAAM representations have been *detectors* and *decoders*, but it is also possible to build *transformers* which directly convert RAAM encoded sentence representations into modified RAAM encoded representations [Chalmers, 1990a]. Again, it must be emphasized that these transformations act directly on the subsymbolic forms—no decoding is done to accomplish the task. Decoding is performed afterwards only to test the accuracy of the transformation. The transformation task chosen for this experiment was to convert encoded representations of the form *X chase Y* directly into encoded representations of the form *Y flee X*. This transformation is considered syntactic because a symbolic system need only apply syntactic rules to accomplish the task; no outside information about the words or their relationships is needed.

For these experiments it was necessary to train a new RAAM with a new data set that included sentences of the appropriate type. A new corpus was created with 20 sentences containing the verb *chase*, the 20 corresponding sentences containing the verb *flee*, and 110 miscellaneous sentences. The RAAM architecture from Figure 8 was used. The RAAM was trained for 3,700 trials on the new 150 sentences. Then every compositional representation containing the word *chase* or *flee* was separated from the rest. In addition, 4 new *chase* sentences and the corresponding *flee* sentences were also encoded in the RAAM and saved.

Sixteen of the RAAM-trained patterns that were formed by encoding the *chase* sentences were then trained to be associated via the feed-forward network (shown in Figure 14) to the sixteen corresponding *flee* encoded patterns. The feed-forward network quickly mastered the task in approximately 75 trials.

After training, the feed-forward network was presented with 8 novel sentences, 4 of which

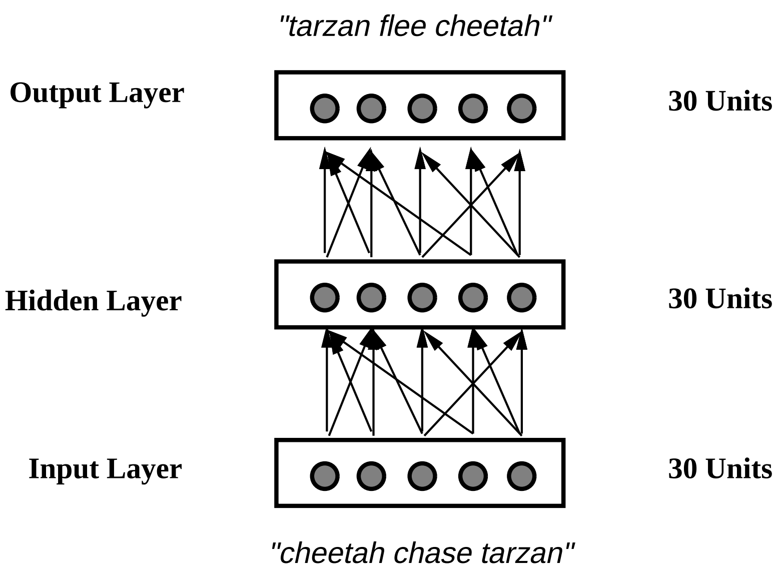

Figure 14: **Feed-forward network architecture for transformations:** The network was trained to take an encoded *chase* sentences as input and to produce the corresponding encoded *flee* sentence as output.

had been trained in the RAAM and 4 of which had not. The transformation network successfully generalized the correct *flee* sentence representations 100% of the time for the RAAM-trained input and 75% of the time for the doubly novel input (novel to the RAAM and to the transformer). The one error made by the network was only on a single word; it transformed *junglebeast chase chimp* into *chimp flee cheetah.* The accuracy of the transformation was checked by taking the resulting output from the feed-forward network and decoding it in the trained RAAM. Figure 15 shows the various steps that were necessary to accomplish this experiment. First the sentence to be transformed was encoded with the trained RAAM. Then the composed representation of the entire sentence was given as input to the transformation network and the network converted this composed structure directly into another composed sentence structure. Finally this output from the transformation network was decoded with the trained RAAM.

The transformation experiment further demonstrates that subsymbolic representations of symbol structures can be operated on in a holistic fashion that is not possible in traditional symbolic systems.

7 Conclusions

In this chapter we have characterized the symbolic and subsymbolic paradigms as two opposing corners of an abstract space of paradigms. This space, we propose, has at least three dimensions: representation, composition, and functionality. By defining the differences in these terms, we are able to place actual models in the paradigm space, and compare and contrast these models in somewhat common terms.

RAAM lies in the subsymbolic portion of the space. We have examined in detail the

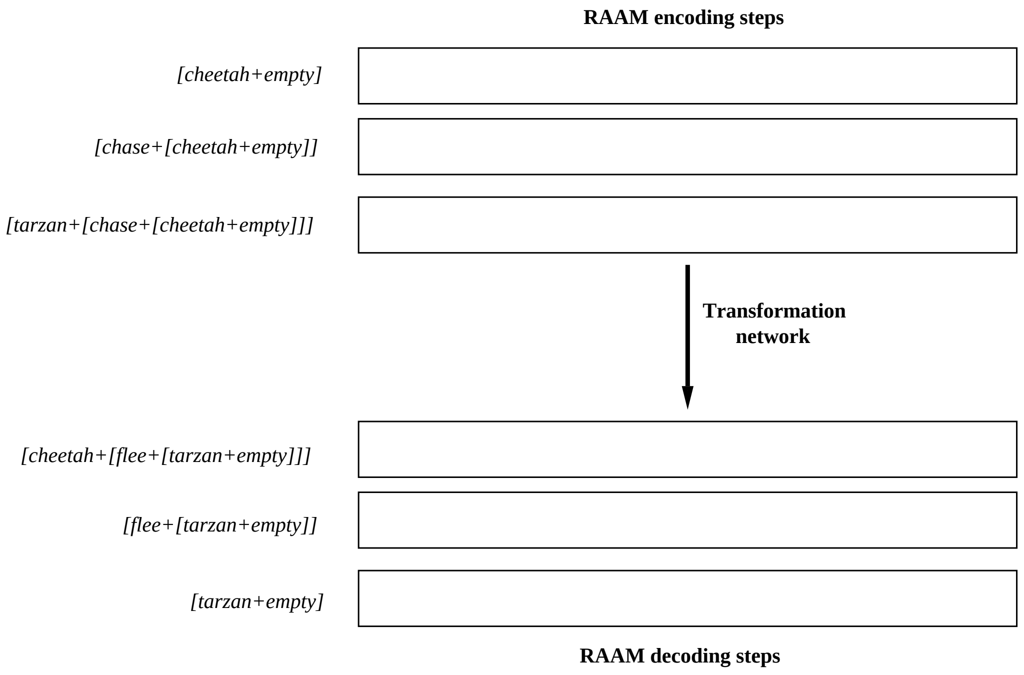

Figure 15: **Holistic transformation process:** This diagram shows each of the steps in the transformation process: encoding a *chase* sentence, holistically transforming the *chase* sentence into the corresponding *flee* sentence, and decoding the *flee* sentence.

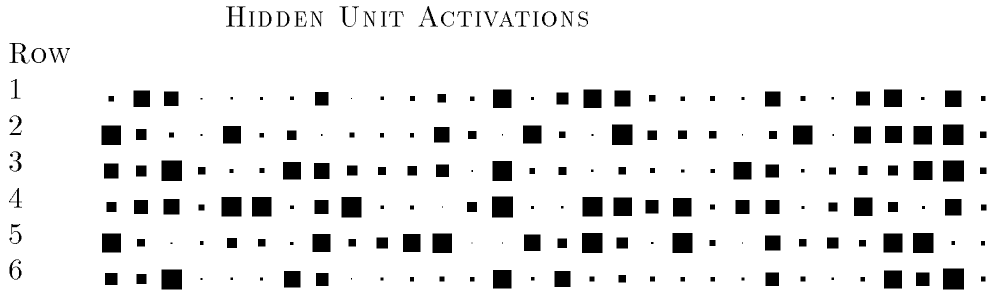

Figure 16: **Temporary figure:** These representations of activations belong in the boxes in Figure 15. Row 1 is the representation of *cheetah + empty*; Row 2 is *chase + cheetah + empty*, etc.

RAAM architecture, representations, compositional mechanisms, and functionality. In conjunction with other simple feed-forward networks, we have exhibited detectors, decoders and transformers which act holistically on the composed, distributed, continuous subsymbolic symbols created by RAAM. These tasks are accomplished without the need to decode composite structures into their constituent parts, as symbolic systems must do.

The RAAM model, developed quite recently, has extended the functionality of the subsymbolic paradigm a great deal. Pollack, the inventor of RAAM, compares RAAM's representations to those of classical symbolism:

> Like feature-vectors, they are fixed-width, similarity-based, and their content is easily accessible. Like symbols, they combine only in syntactically well-formed ways. Like symbol-structures, they have constituency and compositionality. And like pointers, they refer to larger symbol structures which can be efficiently retrieved. But, unlike feature-vectors, they compose. Unlike symbols, they can be compared. Unlike symbol structures, they are fixed in size. And, unlike pointers, they have content.[7]

In order to achieve the ultimate goal of computationally simulating human cognition, a complex architecture seems warranted which takes advantage of the characteristics of both extremes of the continuum—from the systematic macrosemantics of the symbolic paradigm to the holistic operations and microsemantics of the subsymbolic paradigm. However, many current models reside near the periphery of this paradigm space, staying close to one paradigm in all aspects, or taking advantage of the other paradigm primarily along only one dimension. Recently there has been an increased interest in building hybrid models in order to explore the power achieved by combining the two paradigms [Dyer, 1990] (see also Kwasny in this volume and Lange in this volume). It is our expectation that in the future, as systems become more complex and ambitious, the Gap in the center will gradually be filled with systems which blend the capabilities of both paradigms in an effective way.

8 Acknowledgments

We would like to thank Michael Gasser, Robert Port, John Dinsmore, David Chalmers, John Nienart, Jon Rossie, and Ken Aizawa for their comments on drafts of this paper.

[7][Pollack, 1989, pages 529–30]

References

[Blank, 1990] Blank, D. S. (1990). Differences between representational schemas in connectionism and classical artificial intelligence; or why a rose by a symbolic name might not smell as sweet. In Dinsmore, J., editor, *Proceedings of the Midwest Artificial Intelligence and Cognitive Science Society.*

[Chalmers, 1990a] Chalmers, D. J. (1990a). Syntactic transformations on distributed representations. *Connection Science*, 2:53–62.

[Chalmers, 1990b] Chalmers, D. J. (1990b). Why Fodor and Pylyshyn were wrong: The simplest refutation. In *Proceedings of the 12th Annual Conference of the Cognitive Science Society*, pages 340–347.

[Dyer, 1990] Dyer, M. G. (1990). Distributed symbol formation and processing in connectionist networks. *Journal of Experimental and Theoretical Artificial Intelligence*, 2:215–239.

[Elman, 1990] Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14:179–212.

[Fodor and Pylyshyn, 1988] Fodor, J. A. and Pylyshyn, Z. (1988). Connectionism and cognitive architecture: a critical analysis. *Cognition*, 28:3–71.

[Franklin and Garzon, 1990] Franklin, S. and Garzon, M. (1990). Neural computability. In Omidvar, O., editor, *Progress in Neural Networks*, volume 1. Ablex, Norwood, NJ.

[Hofstadter, 1984] Hofstadter, D. R. (1984). The Copycat project: An experiment in nondeterminism and creative analogies. *AI Memo*, 755. Artificial Intelligence Laboratory, MIT, Cambridge, MA.

[Hofstadter and Mitchell, 1991] Hofstadter, D. R. and Mitchell, M. (1991). An overview of the Copycat project. To appear in Holyoak and Barnden, editors, *Connectionist Approaches to Analogy, Metaphor and Case-Based Reasoning.* Ablex.

[Pollack, 1988] Pollack, J. B. (1988). Recursive auto-associative memory: Devising compositional distributed representations. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, pages 33–39. Erlbaum.

[Pollack, 1989] Pollack, J. B. (1989). Implications of recursive distributed representations. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems I*, pages 527–536. Morgan Kaufmann.

[Port and van Gelder, 1991] Port, R. and van Gelder, T. (1991). Dimensions of difference: Compositional representation in AI and connectionism. To be presented at the Spring Symposium Series, Stanford University, under the topic: Connectionist Natural Language Processing.

[Rumelhart et al., 1986] Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning internal representations by error propagation. In McClelland, J. and Rumelhart, D., editors, *Parallel Distributed Processing*, volume I, pages 318–362. MIT Press, Cambridge, MA.

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

[Touretzky, 1990] Touretzky, D. S. (1990). BoltzCONS: Dynamic symbol structures in a connectionist network. *Artificial Intelligence*, 46:5–46.

[van Gelder, 1990] van Gelder, T. (1990). Compositionality: A connectionist variation on a classical theme. *Cognitive Science*, 14.