A common framework for distributed representation schemes for compositional structure

Tony A. Plate
School of Mathematical and Computing Sciences
Victoria University of Wellington, Wellington, New Zealand
Tony.Plate@vuw.ac.nz

Abstract

Over the last few years a number of schemes for encoding compositional structure in distributed representations have been proposed, e.g., Smolensky's tensor products, Pollack's RAAMs, Plate's HRRs, Halford et al's STAR model, and Kanerva's binary spatter codes. All of these schemes can placed in a general framework involving superposition and binding of patterns. Viewed in this way, it is often simple to decide whether what can be achieved within one scheme will be able to be achieved in another. Furthermore, placing these schemes in a general framework reveals unexplored regions in which other related representation schemes with interesting properties.

1 Introduction

Distributed representations are one of the most compelling ideas in connectionism. The use of distributed representations has endowed many connectionist models with their intriguing properties: ability to learn, parallel processing, soft capacity limits, and fault tolerance. However the difficulty of representing compositional structure in a distributed fashion has been a severe obstacle to constructing distributed connectionist models of higher-level reasoning. The basic problem is how to *bind* entities together. For example, in order to reason about a sentence (or concept) such as "John ate the fish", the various entities involved ("John", "the fish", and the "ate" relation) must be bound together in such a way as to preserve the role-filler relations. Merely superimposing the representations of the entities fails to distinguish between "John ate the fish" and "The fish ate John". Although performing single-level binding is straightforward, the problem of representing nested compositional structure is much more difficult. To represent nested compositional structure requires the ability to perform recursive bindings. For example, in "Peter knew that John ate the fish", "John ate the fish" must now be treated as an individual entity involved in a "know" relation. Hinton [1990] discusses this problem and proposes the idea of a "reduced" representation for a structure: a reduced representation is a pattern which (a) acts as a pointer to a structure, and which (b) expresses some information about the content of the structure, and which (c) is the same size as the pattern for a simple object and can be manipulated in the same way.

Appeared in *Connectionist Systems for Knowledge Representation and Deduction*, Frederic Maire, Ross Hayward and Joachim Diederich, eds, pp15–34, Queensland University of Technology, 1997.

In the last few years people have proposed a number of schemes for distributed representation of nested compositional structure. Among these are Smolensky's [1990] Tensor product representations, Pollack's [1990] Recursive Auto-associative Memories (RAAMs), Touretzky's [1990] BoltzCONS, Murdock's [1987, 1993] TODAM model, Plate's [1991, 1994a] Holographic Reduced Representations (HRRs), Sperduti's [1994a] Labeling RAAMs (LRAMMs), and Kanerva's [1996] binary spatter-codes. Although all of these schemes appear quite different on the surface, they can all be viewed in terms of a general framework involving two fundamental operations on patterns. The first is superposition: collections of entities are represented by superpositions of patterns. The second is binding: bindings between entities are represented by some sort of vector-space multiplication of patterns. In contrast to conventional neural-network associative memories, associations between patterns (i.e., bindings) have the same status to patterns; they are represented by activation values over units rather than connection weights between units. This is necessary in order to allow representations involving associations and to allow reasoning about associations. Various schemes also add further operations to these two basic ones, such as non-linearities in RAAMs, and error-correcting auto-associative memory in HRRs, to achieve various abilities. Viewed within this framework, it is often simple to decide whether what can be achieved within one scheme will be able to be achieved in another. Furthermore, placing these schemes in a general framework reveals unexplored regions which might contain other related representation schemes with interesting and useful properties.

There are other methods for storing information about conceptual structure in networks, e.g., the schemes described by Hummel and Biederman [1992] and Shastri and Ajjanagadde [1993] for representing bindings through temporal synchrony, However, storing structures as distributed representations seems particularly promising because it allows one to exploit the ability of neural networks to manipulate and learn flat vector representations, and to exploit the rich similarity structures which are possible with distributed representations [Plate 1994a].

Section 2 discusses the properties of distributed representations in more detail. Section 3 lays out the general framework for viewing schemes for distributed representation of nested compositional structure. Section 4 explains how the various schemes fit into this framework. Section 5 discusses specific and general properties of the schemes and whether or not these properties transfer. Section 6 concludes this paper with a brief discussion of what other schemes one might find in this general framework.

2 Distributed representations

Connectionist representations come in two flavours – local and distributed. Each involves different trade-offs of the properties of the representation space. For models involving only a small number of features and simple items, local representations can be more parsimonious. Distributed representations come into their own for models in which items have features from a very large pool or have complex structure.

In local representations each unit indicates the strength, or merely presence or absence, of

a particular entity. In distributed representations entities are represented by patterns over the units rather than by single units [Hinton, McClelland, and Rumelhart 1986]. Each unit participates in the representation of many entities, and each entity is represented collectively by many units.

In some local representations, individual units correspond to particular features of items. Although these types of local representations can appear very similar to distributed representations when features are numerous and fine-grained, and although local representations can be mapped to distributed ones by a simple linear map, and back by a thresholded linear map, there is still one difference very pertinent to scaling and to the representation of structure. It is that in local representations the total number of possible features is limited to the number of units. This is of vital importance when it comes to representing compositional structure. As Wharton, Holyoak, Downing, Lange, Wickens, and Melz [1994] acknowledge, local representations could be augmented with combinatorial features to represent all possible bindings but note that this would result in a combinatorial explosion in the number of units required when applied recursively to represent things such as the sour-grapes feature: "thing that is desired but can't be obtained and hence is denigrated".

Distributed representations can escape this limitation on the number of features by taking advantage of the property of high-dimensional vector spaces that there are exponentially many more nearly-independent directions than dimensions. Each of these nearly-independent directions can represent a feature, and thus hundreds of thousands of "virtual" features can be represented in a vector with far fewer elements. This involves two-tradeoffs. The first is that only relatively few features can be present at once. However, this is usually the case when there are large numbers of potential features. The second is that there is a chance of interference (ghosting or cross-talk) when too many features are present at once. The probability of interference can be made arbitrarily low by limiting the number of features present at once and choosing a large enough dimension of the vectors.

In summary, the useful and interesting properties of distributed representations include the following:

- Distributed representations offer an efficient use of representational resources (provided that one only wants to represent a small number of entities at any one time, and that some degree of overlap is permissible), because the number of possible patterns over a collection of units is far greater than the number of units in the collection.
- Distributed representations are continuous, or at least fine-grained. This allows nuances of meaning to be expressed by small changes in patterns, and supports learning.
- Distributed representations can be processed in fine-grained parallel networks of simple processing units (i.e., neural networks). Furthermore it is possible to learn or modify patterns by making small changes to them with gradient-based techniques for learning in neural networks.
- Distributed representations can be noise tolerant, as patterns are still recognizable when slightly corrupted.

- Patterns can be superimposed, but still individually recognizable. As more patterns are superimposed, it becomes more difficult to determine which patterns are present. As a result, connectionist memories which use distributed representations tend to have soft capacity limits.
- Using distributed representations it is possible to capture a rich similarity structure over entities, in which surface similarity of patterns is sensitive to both structure and non-structural features of entities (see Gentner and Markman [1993] and Plate [1994a] for further discussion).

3 A common framework

All the abovementioned schemes for representing compositional structure in a distributed fashion can be viewed within a common framework involving two fundamental operations: superposition of patterns and binding of patterns. Different schemes augment these with various other operations in order to achieve particular abilities.

In talking about representations, it is convenient to regard the activation values on a collection of units as a vector, and particular instantiations of those vector values as patterns. Activation values can be either binary, real, or complex. In the following $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ will be used as examples of patterns[1], where A_i etc. are the individual values, and n is the number of elements in the pattern. *Similarity* of patterns refers to either their overlap (in the case of binary patterns) or their inner (dot) product, i.e., $\sum_{i=0}^{n-1} A_i B_i$.

The superposition operation is used to represent unstructured collections of entities. For binary patterns, the elementwise OR is generally used. For patterns of non-binary elements, the elementwise sum is generally used, possibly followed by some transformation that keeps individual values or overall strength of the pattern within some range. Obviously, the superposition of two patterns is the same type of object as each individual pattern — it is still a vector of n values — although the it may have different statistics (e.g., higher density). The essential property of the superposition operation is that it preserves similarity in an unstructured fashion: $\mathbf{A} + \mathbf{B}$ is similar to $\mathbf{A}$ and $\mathbf{B}$, and $\mathbf{A} + \mathbf{B}$ is similar to $\mathbf{A} + \mathbf{C}$.

The binding operation is used to represent bindings or associations between entities. The symbol "$*$" is used to denote a binding operation in this paper, so $\mathbf{A} * \mathbf{B}$ is the binding of the patterns $\mathbf{A}$ and $\mathbf{B}$, and $\mathbf{A} * \mathbf{B}$ is itself a pattern. Quite a variety of operations are used for binding, although all can be regarded as some type of vector multiplication operation. Furthermore, all can be described as a sum of certain elements of the outer product of the patterns (i.e., the n^2 scalar products $A_i B_j$). Smolensky [1990] generalizes this idea with tensor products to bindings involving more than two patterns, e.g., $\mathbf{A} * \mathbf{B} * \mathbf{C}$. In this binding, there are n^3 elements in the corresponding tensor product, i.e., $A_i B_j C_k$. The binding operation is generally more complex than the superposition operation and has several essential properties:

[1] Bold-face names, e.g., $\mathbf{x}$ are used for vectors (patterns).

- it randomizes unstructured similarity: $\mathbf{A} * \mathbf{B}$ is not similar to $\mathbf{A}$ or $\mathbf{B}$;
- it preserves structured similarity: $\mathbf{A} * \mathbf{B}$ is similar to $\mathbf{A}' * \mathbf{B}'$ to the extent that $\mathbf{A}'$ is similar to $\mathbf{A}$ and $\mathbf{B}'$ is similar to $\mathbf{B}$; and
- an inverse (decoding) operation exists, so that the pattern $\mathbf{B}$ can be reconstructed given $\mathbf{A} * \mathbf{B}$ and $\mathbf{A}$ (or vice-versa).

Binding operation differ more than superposition operations. The two the most important dimensions on which binding operations differ are the number of units required to represent $\mathbf{A} * \mathbf{B}$ pattern and the degree of noise in reconstructed patterns. When the $\mathbf{A} * \mathbf{B}$ pattern is the same size as the $\mathbf{A}$ and $\mathbf{B}$ patterns, representations for nested structure is simple to implement. To a large extent, there is an inverse relation between these two dimensions (as would be expected from considerations of information): more accurate reconstructions of components are possible with binding operations for which the pattern is larger. The algebraic properties of binding operations can also vary: some are commutative (i.e., $\mathbf{A} * \mathbf{B} = \mathbf{B} * \mathbf{A}$) and some are not, and some are associative (i.e., $(\mathbf{A} * \mathbf{B}) * \mathbf{C} = \mathbf{A} * (\mathbf{B} * \mathbf{C})$) and some are not.

Binding can be used to represent predicate structure in different ways. One of the most common is role-filler binding, in which predicate structure is represented by the superposition of bindings between role and filler patterns. For instance, the predicate structure of "John ate the fish", can be represented as: $\mathbf{eat\_agent} * \mathbf{john} + \mathbf{eat\_object} * \mathbf{the\_fish}$. A alternative to this is used in Halford, Wilson, Guo, Gayler, Wiles, and Stewart's [1994] STAR model: predicate arguments are bound together, e.g., $\mathbf{eat} * \mathbf{john} * \mathbf{the\_fish}$ (this requires a non-commutative binding operation to avoid confusion with "The fish ate John").

Some the operations used for binding originated in associative memory models (Willshaw, Buneman, and Longuet-Higgins [1969] discuss associative memories based both on the convolution operation and on the outer-product operation), although in that associations are usually represented in connection weights and cannot be manipulated as entities in their own right.

Within this general framework, the various ways in which schemes for representing compositional structure can differ include the following:

1. the nature of the distributed representation: the domain of activation values (binary, real, or complex), and the statistical distribution of pattern values; whether representations are fully or partially distributed;
2. the choice of superposition operation;
3. the choice of binding operation (which determines whether or not bindings are the same type of representational objects as patterns);
4. how the binding operation is used to represent predicate structure (role-filler binding is a common method, but there are others); and

5. the use of other operations and techniques, such as non-linearities, clean-up memory, normalization, and learning.

Why combinatorial features are practical with distributed representations

In distributed representation schemes, the set of features of an entity is often represented by the superposition of those features (in the same way that a collection of entities can be represented by their superposition). The way in which the various schemes represent compositional structure can be understood in the same terms, but with the use of combinatorial features. Recall that the use of combinatorial features such as "thing that is desired but can't be obtained and hence is denigrated" is problematic with local representations because it seems to call for an exponential number of compositional-feature units. However, with distributed representations, an exponential number of virtual features (i.e., patterns) is available. The binding operation provides a systematic way of constructing and decoding the patterns corresponding to compositional features. For example, in the example of the role-filler binding for "John ate the fish", if **person** is a feature of **john** (and thus **john** is similar to **person** by the non-structured similarity-preserving property of superposition), then **eat_agent** $*$ **person** will be a feature of **eat_agent** $*$ **john** $+$ **eat_object** $*$ **the_fish** (by the structured similarity-preserving property of binding). Thus, the pattern for this proposition will be similar to that for "Peter ate the fish", since they both share the **eat_agent** $*$ **person** feature.

4 How various schemes fit in the framework

This section explains how various schemes fit into the framework. It can be skipped by the reader not interested in detail. The choices involved in different schemes are summarized in Table 1. One notable commonality of all of these schemes is that the binding operation is the sum of a set of products, i.e., an element of the pattern $\mathbf{A} * \mathbf{B}$ is the sum of some $A_i B_j$.

4.1 Holographic Reduced Representations

HRRs are a distributed representation for recursive propositional (or frame-like) structures. All entities — objects, predicate labels, and roles — are represented by n-dimensional patterns of real numbers, whose elements have a Gaussian distribution with mean 0 and variance $1/n$. Propositions are encoded as the superposition (sum) of role-filler bindings. A binding is the circular convolution (denoted by $\circledast$) of a role and a filler. The circular convolution of two n-dimensional patterns $\mathbf{x}$ and $\mathbf{y}$ ($\mathbf{z} = \mathbf{x} \circledast \mathbf{y}$) is n-dimensional and has elements

$$z_i = \sum_{k=0}^{n-1} x_k y_{j-k}$$

where subscripts are modulo-n. Circular convolution can be considered as a multiplication operation for vectors, and has algebraic properties in common with scalar and with matrix

Scheme	Patterns	Superposition	Binding	Structure	Other operations
HRRs	real (Gaussian)	addition	circular convolution*	role-filler	normalization to unit-length vectors, clean-up memory, chunking
Frequency-Domain HRRs	complex (unit-circle)	normalized addition	elementwise multiplication*	role-filler	normalization to values on unit-circle, clean-up memory, chunking
Binary spatter codes	dense binary	majority (ties broken randomly)	elementwise XOR*	role-filler	clean-up memory, chunking
TODAM2	real	addition	convolution	paired items, chaining	closest match on output
CHARM	real	addition	truncated convolution*	paired items	closest match on output
Tensor products	real or binary	addition or binary-OR	tensor product	role-filler	
STAR	real or binary	see text	tensor product	argument-product	chunking — see text
RAAMs	real and binary	addition	matrix-vector multiplication	role-filler	sigmoid non-linearities, learning
LRAAMs	real and binary	addition	matrix-vector multiplication	role-filler, labels	sigmoid non-linearities, learning

Table 1: Choices involved in the various schemes. An asterisk indicates binding operations which create patterns of same size as filler patterns, allowing simple construction of nested (recursive) structures.

multiplication: it is associative, commutative, distributive over addition, an identity vector exists, and most vectors have inverses. Circular convolution can also be computed efficiently via the Fast Fourier Transform in $O(n \log n)$ time (see Press, Flannery, Teukolsky, and Vetterling [1992] for algorithms).

A simple propositional representation of "Spot bit Jane" is

$$\mathbf{P}_{bite} = \langle \mathbf{bite} + \mathbf{bite}_{agt} \circledast \mathbf{spot} + \mathbf{bite}_{obj} \circledast \mathbf{jane} \rangle,$$

where **bite** is a predicate label, $\mathbf{bite}_{agt}$ and $\mathbf{bite}_{obj}$ are roles, and **jane** and **spot** are fillers. The angle brackets $\langle \cdot \rangle$ indicate that the pattern is normalized to have a Euclidean length of one (thus individual elements of the pattern still have variance $1/n$). Bindings do not get confused when added together: $\mathbf{bite}_{agt} \circledast \mathbf{spot} + \mathbf{bite}_{obj} \circledast \mathbf{jane}$ is quite distinct from $\mathbf{bite}_{agt} \circledast \mathbf{jane} + \mathbf{bite}_{obj} \circledast \mathbf{spot}$. Even though some components are associations (the bindings) and some are objects (the predicate label), they are all n-dimensional patterns and can be superimposed. Thus, $\mathbf{P}_{bite}$, the representation for the entire proposition, is also an n-dimensional pattern. This makes it simple to use propositions as fillers for roles in other propositions. For example, in the proposition for "Spot bit Jane, which caused Jane to run away from Spot." ($\mathbf{P}_{cause}$, below), $\mathbf{P}_{bite}$ fills the antecedent role of the top-level cause proposition:

$$\begin{aligned} \mathbf{P}_{bite} &= \langle \mathbf{bite} + \mathbf{bite}_{agt} \circledast \mathbf{spot} + \mathbf{bite}_{obj} \circledast \mathbf{jane} \rangle \\ \mathbf{P}_{flee} &= \langle \mathbf{flee} + \mathbf{flee}_{agt} \circledast \mathbf{jane} + \mathbf{flee}_{from} \circledast \mathbf{spot} \rangle \end{aligned}$$

$$\begin{aligned}\mathbf{P}_{objects} &= \langle \mathbf{jane} + \mathbf{spot} \rangle \\ \mathbf{P}_{cause} &= \langle \mathbf{cause} + \mathbf{P}_{objects} + \mathbf{P}_{bite} + \mathbf{P}_{flee} \\ &\quad + \mathbf{cause}_{antc} \circledast \mathbf{P}_{bite} + \mathbf{cause}_{cnsq} \circledast \mathbf{P}_{flee} \rangle\end{aligned}$$

The unbound representations for the objects ($\mathbf{P}_{objects}$) and lower-level HRRs ($\mathbf{P}_{bite} + \mathbf{P}_{flee}$) are included in $\mathbf{P}_{cause}$ in order to make the representation for a proposition have some similarity to those objects and other propositions composed of those objects; these components are not necessary for the representation of the structure.

The distributed representations for the base-level components, i.e., objects, roles, and labels, are based on random patterns of independently distributed elements with mean zero and variance $1/n$. Representations of similar entities (e.g., **jane** and **fred**) are composed of a base type (**person**) and unique random identifying patterns ($\mathbf{id}_{jane}$ and $\mathbf{id}_{fred}$):

$$\begin{aligned}\mathbf{jane} &= \langle \mathbf{person} + \mathbf{id}_{jane} \rangle \\ \mathbf{fred} &= \langle \mathbf{person} + \mathbf{id}_{fred} \rangle.\end{aligned}$$

The mixing proportions of the various components can be changed to make objects of the same type more or less similar. Binding and superposition preserve the Gaussian distribution of elements (modulo scaling or normalization). Thus, all patterns have the same characteristics.

Like the binding operations used in other schemes, convolution is randomizing and similarity preserving. Convolving a role and a filler effectively randomizes with respect to the role and the filler: the binding will not be similar to either the role or the filler (except in special cases). On the other hand, convolution preserves similarity between bindings with similar components: if **jane** is similar to **fred** to some degree then $\mathbf{jane} \circledast \mathbf{role}$ will be similar to $\mathbf{fred} \circledast \mathbf{role}$ to approximately the same degree.

Decoding and clean-up memory

Bindings can be decoded by convolving with the approximate inverse[2] of the role (or filler) pattern. For example, the filler in $\mathbf{bite}_{agt} \circledast \mathbf{spot}$ can be found by convolving it with $\mathbf{bite}^*_{agt}$ ($\mathbf{x}^*$ denotes the approximate inverse of $\mathbf{x}$). The approximate inverse of a vector under convolution is simple to compute: it is a permutation of elements: $\mathbf{x}^*_i = \mathbf{x}_{n-i}$, where subscripts are modulo-n (this is equivalent to reversal of the elements followed by rotation by one position). Thus, like binding, decoding (without cleanup) can be computed in $O(n \log n)$ time.

The result $\mathbf{bite}^*_{agt}\mathbf{bite}_{agt} \circledast \mathbf{spot}$ is a noisy version of **spot**, which must be cleaned up using some sort of auto-associative memory in which the possible fillers have been stored. This clean-up memory must perform closest-match retrieval and must store all patterns, both objects and structures, which can result from decoding a HRR in the system. When

[2]The exact inverse of a pattern under convolution usually exists, but is numerically unstable except under certain conditions. Thus it is usually preferable to use the approximate inverse.

multiple bindings are superimposed, as in $\mathbf{P}_{bite}$, the other bindings do not interfere with the decoding other than by making it more noisy. This is because the randomizing effects of convolution result in convolution products like $\mathbf{bite}^*_{agt} \circledast \mathbf{bite}_{obj} \circledast \mathbf{jane}$ being unrelated to any other pattern[3].

The clean-up memory can also be used to improve performance in decoding nested structure by cleaning up intermediate results. For example, the agent of the antecedent of $\mathbf{P}_{cause}$ can be decoded directly by $\mathbf{bite}^*_{agt} \circledast \mathbf{cause}^*_{antc} \circledast \mathbf{P}_{cause}$ (presuming one knows that the appropriate agent role of the antecedent is $\mathbf{bite}_{agt}$). It can also be decoded using intermediate clean up, first as $\mathbf{cause}^*_{antc} \circledast \mathbf{P}_{cause}$, which is cleaned up to give $\mathbf{P}_{bite}$ (presuming $\mathbf{P}_{bite}$ was stored in the clean-up memory), and next as $\mathbf{bite}^*_{agt} \circledast \mathbf{P}_{bite}$.

Associative memory is used at two levels in HRRs: a noisy but compact associative operation (convolution) is used to represent structure in items, and a error-correcting associative memory is used to store those individual items. The individual items can be regarded as "chunks", and breaking very large structures in structured items of manageable size can be regarded as "chunking". Using intermediate clean-up and chunking, structures of any size can be represented with HRRs.

Scaling properties of HRRs

HRRs require high-dimensional patterns to work well – even small toy problems require patterns with 512 or $1,024$ elements.[4] However, this is not as much of a drawback as it might seem, for two reasons. One reason is that the scaling is very good, thus larger problems do not require much higher dimensionality – tens or hundreds or thousands of objects can be represented using patterns with $2,048$ or $4,192$ elements. The other reason is that the HRR construction and decoding operations are very fast: binding and decoding can be computed in $O(n \log n)$ time. Clean-up (i.e., finding the closest matching pattern in memory) is the only slow operation, but is highly amenable to parallel implementation.

There are two constraints which determine the scaling behaviour of HRRs with respect to capacity (see Plate [1994a] for details):

- The number of components which can be included in one chunk scales linearly with the vector dimension n.
- The number of chunks which can be stored in clean-up memory scales exponentially with n.

The first constraint tends to be the limiting one, so chunks must be kept to a reasonable size. Note that the "number of components" in a HRR depends on the chunking and decoding

[3] Random patterns can be similar by chance, but the probability can be made arbitrarily low with large enough n

[4] Patterns here have dimensions which are powers of 2 only because this makes FFT implementation of convolution simple and efficient.

strategies used. The only components which count towards the size of a chunk are those which must be directly decoded without intermediate cleanup: components inside chunks do not count. Thus, chunking allows structures of unlimited depth to be represented in HRRs.

For toy problems, HRRs seem to require very large patterns. For example, in the structural-similarity experiments involving twelve episodes like "Fido bit Jane, causing Jane to flee from Fido" [Plate 1994a], the patterns had $2,048$ elements. However, in these experiments the limiting factor on capacity was the number of components in an item – with this dimensionality many thousand item can be stored in clean-up memory. A toy problem with a large number of objects in memory in given in Plate [1994a]. The task involves memorizing times and addition tables for numbers from 1 to 50 and retrieving them based on partial cues. Just over $5,000$ items were stored in memory ($2,500$ numbers, and just over $2,500$ relations). The pattern dimension was 512, and no errors were made in over $10,000$ trials of retrieving and decoding a relation based on partial information. The pattern dimension was low compared with the structural similarity task because items only had a maximum of 3 components.

4.2 Frequency-domain (complex-valued) HRRs

In frequency-domain HRRs [Plate 1994b] the values in the distributed representations are complex numbers on the unit circle (with a uniformly distribution of angles). The binding operation is elementwise multiplication, which is the equivalent of circular convolution in the frequency domain of the Fourier transform (see Press, Flannery, Teukolsky, and Vetterling [1992] for algorithms). Superposition is implemented by addition followed by magnitude normalization to make all the values fall on the unit circle. Both superposition and binding clearly result in pattern of the same size, and also preserve the uniform distribution of angles. Thus, representing nested structure composition is straightforward. All the remaining techniques and properties of ordinary HRRs transfer with ease — clean-up memory, role-filler binding, construction of recursive structure, structure matching, etc.

4.3 Binary spatter codes

Kanerva's [1996] binary spatter codes work over binary distributed representations (i.e., bit vectors). The bit vectors are high-density, i.e., the bits are 0 or 1 with equal probability. The binding operation is bitwise exclusive-OR. The superposition operation is the bitwise majority function, with ties broken randomly (ties are only possible when an even number of patterns are superimposed). Both these operations preserve the approximately equal density of 0's and 1's. Thus, representing nested structure is straightforward. This scheme is equivalent to frequency-domain HRRs in which all values are restricted to $+1$ or -1. All the other techniques and properties of ordinary HRRs can be expected to transfer to binary spatter codes. Although much noise can be introduced by the superposition operation (in the superposition of two patterns half the bits will be random), the scheme has sufficient capacity to be interesting and potentially very useful.

4.4 CADAM, TODAM, TODAM2 and CHARM

Liepa's [1977] CADAM[5] Murdock's TODAM[6] [Murdock 1982] and TODAM2 [Murdock 1993], and Metcalf-Eich's CHARM[7] [1982, 1985] use other forms of convolution to bind items in models of human memory. These models use real-valued patterns to represent items, and use addition for superposition. CADAM, TODAM and TODAM2 use ordinary convolution for binding, which when used to bind two n-element patterns produces a pattern with $2n-1$ elements. CHARM uses the same form of convolution, but only retains the central n elements of the binding pattern.

The emphasis in these models is on retrieval of paired-items and sequences rather than on nested compositional structure. In TODAM, lists of paired items are stored as the superposition of bindings and items, e.g, the pairs **AB** and **CD** are stored in the form $\mathbf{A}*\mathbf{B}+\mathbf{A}+\mathbf{B}+\mathbf{C}*\mathbf{D}+\mathbf{C}+\mathbf{D}$. In CHARM, item information is auto-convolved; the same pairs would be stored as $\mathbf{A}*\mathbf{B}+\mathbf{A}*\mathbf{A}+\mathbf{B}*\mathbf{B}+\mathbf{C}*\mathbf{D}+\mathbf{C}*\mathbf{C}+\mathbf{D}*\mathbf{D}$. These differences result in different access and retrieval properties and consequently in different psychological predictions. The representations in TODAM2 involve the superposition of many more higher-order bindings. The details of these schemes are not important here, save to note that the choice of what to bind and what to store in the trace has important consequences for the properties of the model.

4.5 Smolensky: Tensor products

In tensor product representations [Smolensky 1990], the tensor product is used as the binding operation. The tensor product can be seen as a generalized outer product. Given two n-dimensional column-vectors $\mathbf{x}$ and $\mathbf{y}$, the second-order tensor product $T=\mathbf{x}\otimes\mathbf{y}$ is equivalent to the outer product $\mathbf{x}\mathbf{y}^{\mathrm{T}}$. T has n^2 elements, and $T_{ij}=x_iy_j$. Both lower-order and higher-order tensors exist: a first-order tensor is just a vector, and a zeroth-order tensor is a scalar. A third-order tensor is the tensor product of three vectors: $T=\mathbf{x}\otimes\mathbf{y}\otimes\mathbf{z}$, where $T_{ijk}=x_iy_jz_k$ and T has n^3 elements. Higher-order tensor products are constructed in a similar fashion – a k'th-order tensor is the product of k n-dimensional vectors and has n^k elements. Tensor products can be built up iteratively – multiplying a third-order tensor and a second-order tensor gives a fifth-order tensor. Tensor products can be decoded by taking inner products, e.g., if $T=\mathbf{x}\otimes\mathbf{y}\otimes\mathbf{z}$, then $T\cdot(\mathbf{x}\otimes\mathbf{y})=\mathbf{z}$ (under appropriate normalization conditions on the vectors). The inner product can be taken along any combination of dimensions. For example, given a third-order tensor $\mathbf{x}\otimes\mathbf{y}\otimes\mathbf{z}$, we can take the inner product with a second-order tensor in three different ways: along the first and second dimensions to extract $\mathbf{z}$, along the first and third dimensions to extract $\mathbf{y}$, or along the second and third dimensions to extract $\mathbf{x}$. Care must be taken to use the appropriate inner product.

Tensor products can be used in various ways to represent structured objects. Smolensky

[5] CADAM≡Content addressable distributed associative memory.
[6] TODAM≡Theory of distributed associative memory.
[7] CHARM≡Composite holographic associative recall model.

[1990], Dolan and Smolensky [1989], and Dolan [1989] use role-filler schemes. Nested structures can be represented, but the size of patterns increases exponentially with the depth of nesting. Tensor products bindings are one extreme of using the pairwise products of pattern elements: every possible product is used, and each forms one element of the binding pattern.

It is possible to analyse other binding schemes in terms of tensor products. For example, Dolan and Smolensky [1989] show how Touretzky's [1990] BoltzCONS can be seen as a using subsets of elements of the tensor product. The framework proposed in this paper is to view binding operations as being the sum of elements of the tensor product.

4.6 STAR

Halford, Wilson, Guo, Gayler, Wiles, and Stewart [1994] propose a different way of using tensor products to represent predicates. They want a memory system which can store multiple predicates and retrieve one component of a predicate (the name or a filler) given its remaining components, in order to do analogical reasoning (hence the name "Structured Tensor Analogical Reasoning"). They represent a predicate by the tensor product of all the components of the predicate. The role of a filler determines its position in the product. For example, the representation of `mother-of(woman, baby)` is **mother** $\otimes$ **woman** $\otimes$ **baby**. They represent a collection of predicates by the sum of tensors for individual predicates. This representation allows them to solve analogy problems of the form "Mother is to baby as mare is to what?" For example, suppose T is a sum of predicates, including `mother-of(woman, baby)` and `mother-of(mare, foal)`. The analogy can be solved by first taking the appropriate inner product between the tensors T and **woman** $\otimes$ **baby**,[8] yielding **mother** (provided T does not contain other predicates relating `mother` and `baby`). Having discovered that the relevant predicate is `mother-of` we next take the inner product between **mother** $\otimes$ **mare** and T, yielding the answer **foal**.

Halford, Wilson, Guo, Gayler, Wiles, and Stewart do not focus on how recursive structure might be represented. They do mention chunking as a way of coping with more than four dimensions and with nested predicates, but do not say how vectors representing chunks, which would be akin to reduced representations, might be constructed.

Unlike in HRRs and other schemes, the STAR model uses only one level of associative memory, and does not use superposition in the representation of compositional structure. Superposition is used only to represent a collection of predicates.

STAR is the only scheme discussed here which does not use role-filler binding (Murdock [1983] makes some suggestions for other possibilities, but does not develop them). STAR-like representations could be expected to have quite different properties to role-filler based methods like HRRs and RAAMs with respect to structure matching, structure classification, and structure transformation. For examples, proposition will be different to another if only one of the predicate label or arguments is different, whereas role-filler representations are

[8] Care must be exercised to take the appropriate inner product, which depends upon the position of the unknown component in the third-order tensor.

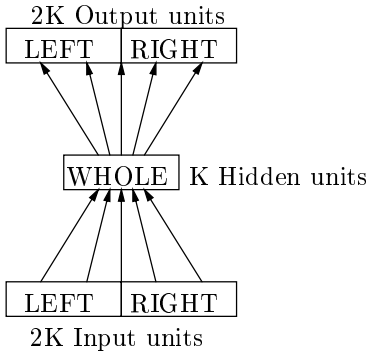

Figure 1: A Recursive Auto-Associative Memory (RAAM) for binary trees. The "WHOLE" is the code for an internal node in a tree, and "LEFT" and "RIGHT" can be codes for either internal or external nodes. (Adapted from Pollack [1990].)

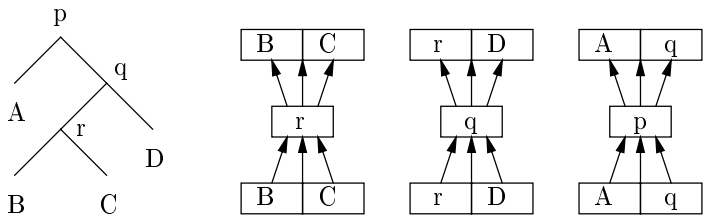

Figure 2: A simple tree and the auto-associations that encode it in a RAAM.

similar if any of the role-filler pairs (or predicate label in the case of HRRs) are similar.

4.7 RAAMs and LRAAMs

Pollack [1990] uses back-propagation to learn reduced representations for trees. He sets up an auto-encoder net to learn to compress the fields of a node to a label, and uncompress the label to the fields. Pollack calls this type of network a Recursive Auto Associative Memory (RAAM).

Figure 1 shows the architecture of the network for binary trees, and Figure 2 shows the three auto-associations the network must learn to encode a simple tree. One network can learn to encode a collection of related trees, e.g., parse trees for the same language, but for simplicity this example shows only a single tree. The codes for terminal nodes (A, B, C, and D) are supplied to the network, but the network must learn suitable codes for the internal nodes (p, q, and r). The training is a moving target problem, because when the weights are adjusted for one example (e.g., for (B C) $\rightarrow$ r $\rightarrow$ (B C)), this changes the representation for r, which changes the target for another example.

A RAAM can be viewed as being composed of two networks – an encoding net (the bottom half), and a decoding net (the top half). The encoding net converts a full representation to a reduced representation, and the decoding net performs the inverse function. The reduced

representation for the tree (B C) is r, the reduced representation for ((B C) D) is q, and so on. The network learns the decoding and encoding functions simultaneously during training. Both encoding and decoding are recursive operations. The encoding procedure knows, from the structure of the tree, how many times it must recursively compress representations. However, the decoding procedure must decide during decoding whether or not a decoded field represents a terminal node or an internal node which should be further decoded. Pollack solves this problem by using "binary" codes for terminals (i.e., each value in the code is 0 or 1). The reduced representations (i.e., codes for internal nodes) developed by the network tend to have values between 0 and 1, but not close to 0 or 1. If a decoded field has all its values sufficiently close to 0 or 1, then it is judged to be a terminal node and is not decoded further.

RAAMs are not limited to binary trees – a RAAM with M input fields (each of K units) can encode trees in which each node has up to M children. Nodes with less than M children can be encoded with special "nil" labels. Each child must appear in a particular place – the left subtree is distinct from the right subtree. The locations of subtrees correspond to roles in predicate structures, so RAAMs can be seen as having a fixed set of roles.

Pollack trained a RAAM with three roles to encode compositional propositions such as (thought pat (knew john (loved mary john))) ("Pat thought that John knew that Mary loved John"). The network has 48 input units, 16 hidden units, and 48 output units. The network learned to store the training set of 13 complex propositions. It is also able to encode and decode some, but not all, of the novel propositions presented to it. Pollack performed a cluster analysis on the codes for trees, which shows that similar trees tended to have similar codes. For examples, the codes for the trees (LOVED JOHN PAT), (LOVED MARY JOHN), (LOVED JOHN MARY), and (LOVED PAT MARY) are all more similar to each other than any of the codes for other trees.

The similarity structure in the codes (reduced representations) indicates that they do provide some explicit information about subtrees (full representations). Pollack probed the nature of this representation by testing whether it is possible to manipulate representations without decoding them. He trained another network to transform reduced descriptions of propositions like (LOVED X Y) to (LOVED Y X), where X and Y can take on four different values. This network has 16 input units, 8 hidden units and 16 output units. Pollack trained the network on 12 of the 16 propositions, and the network correctly generalizes to the other four. Chalmers [1990] explores this further with a network with a RAAM which stores syntactic structures and another network which transforms reduced representations for passive structures to reduced representations for active structures, without decoding.

Sperduti [1994b] proposed a modification of the RAAM network which included a label field (Labeling, or "LRAAMs"). This enables arbitrary graph structure to be stored, rather than just trees. Sperduti and Starita [1995] show how structures can be classified by applying a neural network classifier to LRAAM patterns.

Smolensky [personal communication] observed that RAAMs can be viewed as implementing role-filler binding by matrix-by-vector multiplication. The fillers (input unit activations) are vectors, and the roles (the weights from input to hidden units) are matrices. In fact, this

binding operation is the sum of certain elements of the tensor product of the role matrix and the filler vector. The total input to the hidden units is the superposition of matrix-vector products of roles and fillers. The weights from the hidden to output units must implement the inverse of the role-binding operation, and clean up output vectors as well. The non-linearities on the hidden and output units may help with the clean-up, but the also complicate this interpretation.

5 Properties of the schemes

Given the similarities between the different schemes, we can expect that there will be some properties which are shared by all. The fundamental shared property is that the binding operation generates combinatorial-feature patterns which are used to represent structure, and that distributed representations provide a large enough pattern space with a moderate number of units. Two very important abilities that all the schemes can be expected to have are structure-sensitive matching and classification, and structure transformation. These are discussed in detail in the next subsections.

There are also some techniques which are generally applicable, even if they have not yet been applied to all schemes in the literature. The two most important are chunking using error-correcting auto-associative item memories, and using non-linearities to more closely pack the representational space.

The various differences between the schemes result in some properties that are particular to individual schemes. The most important of these are as follows:

- Bindings are "first class" objects? If $\mathbf{A} * \mathbf{B}$ is the same type of pattern as $\mathbf{A}$ and $\mathbf{B}$ (or at least the same as one of them), then it is straightforward to store nested compositional structure using role-filler bindings. This is the case for HRRs, binary spatter codes, and RAAMs. Chunking is simple to apply when bindings are the same size as the original patterns. However, with tensor products, bindings grow exponentially in size with the depth of structure.
- Is structure represented by role/filler bindings (HRRs, binary spatter codes, tensor products, RAAMs) or by argument-product bindings (STAR)? This has important consequences for the properties of the model.
- Status of role patterns. In RAAMs, role patterns cannot vary (once the RAAM net has been trained), whereas in the other role-filler schemes, role patterns can be different for different predicates. Furthermore, in these other schemes, role patterns can be made similar or different. For example, we might want the role patterns for the agent roles of "bite" and "chew" to be quite similar, and the agent roles of "eat" and "sell" to be quite different. This will have consequences for the structure-sensitive operations.
- Algebraic properties of superposition and binding operations. Ordinary and wrapped convolution are commutative ($\mathbf{A} * \mathbf{B} \equiv \mathbf{B} * \mathbf{A}$) and associative ($(\mathbf{A} * \mathbf{B}) * \mathbf{C} \equiv \mathbf{A} * (\mathbf{B} *$

C)), whereas the tensor product is only associative. This has consequences for how structures are decoded: convolution structures can be decoded in any order, but tensor product structures must be decoded in a particular order. The non-commutativity of the tensor product is essential to the STAR model: if STAR were implemented with a commutative binding operation the order of arguments would be lost. Non-commutative and non-associative versions of convolution do exists, see Plate [1994a].

5.1 Estimating structural similarity by surface comparisons of distributed representations

Gentner and Markman, [Gentner and Markman 1993] expressed skepticism regarding the possibility of development of a connectionist representation in which the superficial features of representations reflect structural composition. It would be very useful if representations were similar to the degree that the structures they represented were similar. Such a representation would allow calculation of structural match by the dot product of vectors. This is a very fast operation and could applications in domains from database indexing to cognitive models of analogical reminding.

It turns out that the superficial features of HRRs can reflect their structural composition. If the entities in structures are similar, then the HRRs are similar to the degree that their structures are similar. This allows the estimation of degree of structural match by the dot product of vectors (HRRs). For example "Spot bit Jane, causing Jane to flee from Spot" is structurally isomorphic to "Fido bit John, causing John to flee from Fido", and the HRRs representing these situations have a higher dot-product than ones representing non-isomorphic situations (see Plate [1994a, 1997] for details). The sensitivity to structure is due to structural features being expressed in the surface form of the pattern in the form of combinatorial feature patterns. While ordinary HRRs are only sensitive to structural similarity when the entities involved are similar, it is a simple to augment HRRs with further combinatorial features which cause them to be sensitive to structural similarity in other situations (this is called "contextualization" in Plate [1994a, 1997]).

In fact, we can expect that in all role-filler binding schemes the superficial features of the patterns will reflect their structural composition. The extent to which they do will depend on whether combinatorial features are transformed in non-linear context-dependent ways. RAAMs and LRAAMs are the only role-filler binding scheme which use non-linear transforms (sigmoid non-linearities on the hidden units). These non-linear transforms are not particularly strong, and it is unclear how much they would change combinatorial features. In shallow structures it would not be surprising if they had little effect and RAAMs had similar structure-matching properties to HRRs. Indeed, the work of Sperduti and Starita [1995] shows that LRAAMs can be used for structure classification, which can be seen as a type of structure matching.

The ability to perform analogical, or structural, matching is not an esoteric, seldom used ability. Rather, it is essential to any reasoning system based on rules or cases. Rule following requires the ability to perform structural matches between a description of some specific situation and an abstract description of a general situation, and case-based reasoning

requires the matching of descriptions at the same level of abstraction. These types of reasoning also require the ability to identify correspondences between descriptions, which can be done with HRRs [Plate 1994a]. The structural classifications described by Sperduti, Starita, and Goller [1997] can be viewed as an example of structural matching between abstract templates and specific instances.

5.2 Transformations without decomposition

One of properties of reduced representations which makes them interesting is the possibility of operating on them without unpacking them. This is a type of fast computation with no obvious parallel in conventional symbol manipulation. Comparison and classification of structures is a simple type of computation without unpacking, and structure transformation is a more complex one. Various authors have demonstrated that structural transformations without decoding can be performed on Pollack's RAAMs and on Smolensky's tensor-product representations. Pollack [1990] trained a feedforward network to transform reduced descriptions for propositions like (LOVED X Y) to ones for (LOVED Y X), where the reduced descriptions were found by a RAAM. Chalmers [1990] trained a feedforward network to transform reduced descriptions for simple passive sentences to ones for active sentences, where again the reduced descriptions were found by a RAAM. Niklasson and van Gelder [1994] trained a feedforward network to do material conditional inference (and its reverse) on reduced descriptions found by a RAAM. This involves transforming reduced descriptions for formulae of the form $(A \rightarrow B)$ to ones of the form $(\neg A \vee B)$ (and vice-versa). Legendre, Miyata, and Smolensky [1991] showed how tensor product representations for active sentences could be transformed to ones for passive sentences (and vice-versa) by a pre-calculated linear transformation. Dolan [1989] showed how multiple variables could be instantiated in parallel, again using a pre-computed linear transformation.

Plate [1994a, 1997] describes how the same kinds of transforms can be performed with HRRs. It is reasonable to expect that at least some transforms can be performed with any of the schemes described, using such operations as transforming the binding $\mathbf{A} * \mathbf{B}$ to $\mathbf{A} * \mathbf{C}$ by binding (multiplying) $\mathbf{A} * \mathbf{B}$ with $\mathbf{B}^{-1} * \mathbf{C}$.

5.3 Chunking

Chunking, i.e., breaking large structures in pieces of manageable size, is readily implementable with any of the schemes for which binding patterns are the same size as the original patterns. Chunking requires the use of an error-correcting auto-associative memory in which all chunks are stored. Chunking makes it possible to store structures of unlimited size. When large structures are decoded or traversed, intermediate results can be cleaned up.

In general, any system that uses chunks must have also have a way of referring, or pointing to the chunks. Schemes like HRRs provide a very attractive way of constructing chunks because a HRR can provide pointer information, but can also provide information about its contents

without dereferencing any pointers. This is what the "reduced" in "Holographic Reduced Representation" refers to – a HRR is a compressed, or reduced, version of the structure it represents. The advantage of having a pointer which encodes some information about what it refers to is that some operations can be performed without following the pointer. This can save much time. For example, nested fillers can be decoded quickly without clean up of intermediate results if very noisy results are acceptable, or the similarity of two structures can be estimated without decoding the structures.

5.4 Non-linearities for packing space

RAAMs appear to use representational space efficiently; published models have worked with quite low dimensional patterns. In contrast, HRRs require relatively high dimensional patterns. This difference can be attributed to the use the learning in RAAMs: they learn how to pack the given structures efficiently into the space. Although this seems like a good thing, there are two costs. The first is that RAAMs do not have good generalization: they have difficulty representing structures other than the ones in the training set. The second is that RAAMs require very high precision values in the patterns, whereas other methods can get by with quite noisy patterns.

Similar learning techniques can also be used with other methods. For example, Plate [1994a] constructed recurrent networks which used a convolution operation for storing sequences. The networks learned to work effectively with much lower dimensional patterns than are needed in a system without learning.

6 Discussion

The general framework for distributed representation schemes for compositional structure provides a common way of analysing and understanding how these schemes represent structure, and what is happening when structures are matched, classified, or transformed. This allows us to recognize commonalities, see which differences are important, and which properties can be transferred.

The framework also reveals unexplored regions where useful schemes might be found. For example, none of the schemes described which nested structure will work very well with sparse binary patterns, because the binding and superposition operations both change the density of sparse patterns. The same goes for sparse real valued patterns (ones in which large values are rare), and non-negative real valued patterns. Finding a scheme that worked with sparse patterns would be interesting because it seems that the human brain probably uses sparse distributed representations [Feldman 1986]. There may be simple modifications of current binding and superposition operations which will work, or there may be other operations which could be used instead. Different binding operations are another unexplored region. In all of the schemes described, the binding operation uses very ordered selections of the elements of the tensor product of the bound patterns. Plate [1994a] shows that random sums of these elements can also be an effective binding operation. The properties of such a

scheme, and how the degree of randomness affects it, have not been investigated.

References

Chalmers, D. J. (1990). Syntactic transformations on distributed representations. *Connection Science 2*(1 & 2), 53–62.

Dolan, C. P. (1989). Tensor manipulation networks: Connectionist and symbolic approaches to comprehension, learning, and planning. Computer Science Department, AI Lab. Technical Report UCLA-AI-89-06, UCLA.

Dolan, C. P. and P. Smolensky (1989). Tensor product production system: a modular architecture and representation. *Connection Science 1(1)*, 53–68.

Feldman, J. A. (1986). Neural representation of conceptual knowledge. Technical report TR189, Department of Computer Science, University of Rochester, Rochester NY.

Gentner, D. and A. B. Markman (1993). Analogy – Watershed or Waterloo? Structural alignment and the development of connectionist models of analogy. In C. L. Giles, S. J. Hanson, and J. D. Cowan (Eds.), *Advances in Neural Information Processing Systems 5 (NIPS*92)*, San Mateo, CA, pp. 855–862. Morgan Kaufmann.

Halford, G. S., W. H. Wilson, J. Guo, R. W. Gayler, J. Wiles, and J. E. M. Stewart (1994). Connectionist implications for processing capacity limitations in analogies. In K. J. Holyoak and J. Barnden (Eds.), *Analogical Connections*, Volume 2 of *Advances in Connectionist and Neural Computation Theory*. Norwood, NJ: Ablex.

Hinton, G. E. (1990). Mapping part-whole heirarchies into connectionist networks. *Artificial Intelligence 46*(1-2), 47–76.

Hinton, G. E., J. L. McClelland, and D. E. Rumelhart (1986). Distributed representations. In D. E. Rumelhart, J. L. McClelland, and the PDP research group (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition.*, Volume 1, pp. 77–109. Cambridge, MA: MIT Press.

Hummel, J. E. and I. Biederman (1992). Dynamic binding in a neural network for shape recognition. *Psychological Review 99*(3), 480–517.

Kanerva, P. (1996). Binary spatter-coding of ordered k-tuples. In C. von der Malsburg, W. von Seelen, J. Vorbruggen, and B. Sendhoff (Eds.), *Artificial Neural Networks–ICANN Proceedings*, Volume 1112 of *Lecture Notes in Computer Science*, Berlin, pp. 869–873. Springer.

Legendre, G., Y. Miyata, and P. Smolensky (1991). Distributed recursive structure processing. In D. Touretzky and R. Lippman (Eds.), *Advances in Neural Information Processing Systems 3*, San Mateo, CA, pp. 591–597. Morgan Kaufmann.

Liepa, P. (1977). Models of content addressable distributed associative memory. Unpublished manuscript.

Metcalfe Eich, J. (1982). A composite holographic associative recall model. *Psychological Review 89*, 627–661.

Metcalfe Eich, J. (1985). Levels of processing, encoding specificity, elaboration, and charm. *Psychological Review 92*, 1–38.

Murdock, B. B. (1982). A theory for the storage and retrieval of item and associative information. *Psychological Review 89*(6), 316–338.

Murdock, B. B. (1983). A distributed memory model for serial-order information. *Psychological Review 90*(4), 316–338.

Murdock, B. B. (1987). Serial-order effects in a distributed-memory model. In D. S. Gorfein and R. R. Hoffman (Eds.), *MEMORY AND LEARNING: The Ebbinghaus Centennial Conference*, pp. 277–310. Lawrence Erlbaum Associates.

Murdock, B. B. (1993). TODAM2: A model for the storage and retrieval of item, associative, and serial-order information. *Psychological Review 100*(2), 183–203.

Niklasson, L. and T. van Gelder (1994). Can connectionist models exhibit non-classical structure sensitivity? In *Proceedings of the Sixteenth Annual Conference of The Cognitive Science Society*.

Plate, T. A. (1991). Holographic Reduced Representations: Convolution algebra for compositional distributed representations. In J. Mylopoulos and R. Reiter (Eds.), *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, San Mateo, CA, pp. 30–35. Morgan Kaufmann.

Plate, T. A. (1994a). *Distributed Representations and Nested Compositional Structure*. Ph. D. thesis, Department of Computer Science, University of Toronto. Available at http://www.cs.utoronto.ca/~tap.

Plate, T. A. (1994b). Estimating structural similarity by vector dot products of holographic reduced representations. In J. D. Cowan, G. Tesauro, and J. Alspector (Eds.), *Advances in Neural Information Processing Systems 6 (NIPS*93)*, San Mateo, CA, pp. 1109–1116. Morgan Kaufmann.

Plate, T. A. (1997). Structure matching and transformation with distributed representations. In R. Sun and F. Alexandre (Eds.), *Connectionist Symbolic Integration*. Lawrence Erlbaum Associates.

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence 46*(1-2), 77–105.

Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling (1992). *Numerical Recipes in C* (Second ed.). Cambridge: Cambridge University Press.

Shastri, L. and V. Ajjanagadde (1993). From simple associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences 16*(3), 417–494.

Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence 46*(1-2), 159–216.

Sperduti, A. (1994a). Encoding of labeled graphs by labeling raam. In J. D. Cowan, G. Tesauro, and J. Alspector (Eds.), *Advances in Neural Information Processing Systems 6*, San Mateo, CA, pp. 1125–1132. Morgan Kaufmann.

Sperduti, A. (1994b). Labeling raam. *Connection Science 6*(4), 429–459.

Sperduti, A. and A. Starita (1995). Supervised neural networks for the classification of structures. Technical Report TR-16/95, University of Pisa, Dipartimento di Informatica.

Sperduti, A., A. Starita, and C. Goller (1997). Distributed representations for terms in hybrid reasoning systems. In R. Sun and F. Alexandre (Eds.), *Connectionist Symbolic Integration*. Lawrence Erlbaum Associates.

Touretzky, D. S. (1990). Boltzcons: Dynamic symbol structures in a connectionist network. *Artificial Intelligence 42*(1-2), 5–46.

Wharton, C. M., K. J. Holyoak, P. E. Downing, T. E. Lange, T. D. Wickens, and E. R. Melz (1994). Below the surface: Analogical similarity and retrieval competition in reminding. *Cognitive Psychology 26*, 64–101.

Willshaw, D. J., O. P. Buneman, and H. C. Longuet-Higgins (1969). Non-holographic associative memory. *Nature 222*, 960–962.