Distributed Representation and Parallel Processing of Recursive Structures

Yoshiro Miyata
Chukyo University

Paul Smolensky
University of Colorado, Boulder

Géraldine Legendre
University of Colorado, Boulder

Abstract

We have developed principles integrating connectionist and symbolic computation by establishing mathematical relationships between two levels of description of a single computational system: at the lower level, the system is formally described in terms of highly distributed patterns of activity over connectionist units, and the dynamics of these units; at the higher level, the same system is formally described by symbolic structures and symbol manipulation. In this paper, we propose a specific treatment of recursion where complex symbolic operations on recursive structures are mapped to massively parallel manipulation of distributed representations in a connectionist network.

1 Integration of Connectionist and Symbolic Computation

This paper concerns the computational backbone supporting the Sub-Symbolic Paradigm research program [Smolensky, 1988]: research studying how symbolic computation can arise naturally as a higher-level virtual machine realized in appropriately designed lower-level connectionist networks.

1.1 The Principles

We begin by briefly presenting the current formulation of two of the fundamental computational principles, one concerning the nature of mental representation and one concerning the nature of mental processing operating on the representation. We then proceed in subsequent sections to present a concrete implementation of these principles. These principles are hypothesized to operate in higher cognitive domains, where cognitive theory has posited symbolic representations that play a central role.

1.1.1 Integrated Representation.

a. When analyzed at the lower level, mental representations are distributed patterns of connectionist activity; when analyzed at a higher level, these same representations constitute symbolic structures.

b. Such a symbolic structure $\mathbf{s}$ is a set of *filler/role bindings* $\{\mathbf{f}_i/r_i\}$, defined by a collection of structural roles $\{r_i\}$ each of which may be occupied by a filler $\mathbf{f}_i$—a constituent symbolic structure.

c. The corresponding lower-level description is an activity vector $\mathbf{s} = \sum_i \mathbf{f}_i \otimes \mathbf{r}_i$ which is the sum of vectors representing the filler/role bindings. In these *tensor product representations*, the pattern for the whole is the superposition of patterns for all the constituents. The pattern for a constituent is the tensor product of a pattern for the filler and a pattern for the structural role it occupies.[1]

[1] The tensor product $\otimes$ generalizes the outer product of matrix algebra. If $\mathbf{u} = (u_1, u_2); \mathbf{v} = (v_1, v_2)$ then their tensor product $\mathbf{u}\otimes\mathbf{v}$ is a *second-rank tensor*, a vector whose elements are normally labelled with *two* subscripts: $(\mathbf{u}\otimes\mathbf{v})_{ij} \equiv u_i v_j$; the elements of $\mathbf{u}\otimes\mathbf{v}$ are thus $(u_1v_1, u_1v_2, u_2v_1, u_2v_2)$. In general, a tensor of rank n has elements labelled with n subscripts, and the tensor product extends in the obvious way to tensors of arbitrary rank; e.g., the tensor product of a rank-2 tensor $\mathbf{S}$ and a rank-3 tensor $\mathbf{T}$ is a rank-5 tensor $\mathbf{R} = \mathbf{S}\otimes\mathbf{T}$ with elements $R_{ijklm} \equiv S_{ij}T_{klm}$. The recursive construction of tensor product representations requires the use of tensors of rank higher than two, which is why simpler matrix algebra does not suffice.

d. In certain cognitive domains such as language and reasoning, the representations are *recursive*: fillers which are themselves complex structures are represented by vectors which in turn are recursively defined as tensor product representations. The set of fillers is then the same as the set of whole structures.

Tensor product representations [Smolensky, 1987b, Smolensky, 1990] generalize many of the traditional kinds of connectionist representations, and while the generic tensor product representations are fully distributed, special cases reduce to semi- or fully local representations.[2] The units in tensor product representations can sometimes be interpreted as the conjunction of a feature of a filler and a feature of its role; in these cases, the approach is a formalization of the idea of *conjunctive coding* [Hinton et al., 1986] Psychological models of human memory have also employed tensor product or closely related representations (e.g., [Anderson et al., 1984]). Tensor calculus (unlike matrix algebra) allows such representations to be defined recursively; in this paper, we will describe an analysis of the recursive capabilities of tensor product representations.

1.1.2 Integrated Processing.

a. When analyzed at the lower level, mental processes consist in massively parallel spreading of numerical activation; when analyzed at a higher level, these same processes constitute a form of symbol manipulation in which entire structures are manipulated in parallel.

b. Certain of these processes can be described precisely in terms of higher level programs. Like traditional computer programs, these programs describe complex functions by sequentially combining primitive symbolic operations. Such programs specify the *input/output function that is computed*, but the complex sequences of primitive operations do *not* constitute procedures by which these functions are actually computed.

c. These processes are capable of fully productive recursive structure processing.

[2]Thus even those who believe that neural representations are not highly distributed should not see this as any objection to the use of tensor product representations as a low-level model of mental representations. If desired, special cases of the tensor product representation can be designed with any desired degree of locality, up to and including representations which dedicate a single node to an entire structure (e.g, proposition).

Structure-sensitive symbolic processing of tensor product representations is achieved by means of operations from tensor calculus which check conditions on constituents and which use linear transformations to move constituents in given structural roles to new ones, or to modify the fillers in given roles. Such operations are naturally embodied in connectionist networks [Dolan and Smolensky, 1989, Legendre et al., 1991, Smolensky, 1987b]. Some new extensions of this principle are developed in this paper.

1.2 An Example Simulation

We have developed the concrete mathematical techniques needed to perform computations using these principles, realized in computer simulations. One simple simulation, which we describe here, was designed purely to demonstrate the formal capabilities of the technique; it takes as input a distributed pattern of activity representing the tree structure underlying an English sentence, determines by inspecting the structure whether the form is that of an active or passive sentence, and, accordingly, produces as output a distributed representation of a tree structure encoding a predicate-calculus form of the semantic interpretation of the input sentence [Legendre et al., 1991]. The network, "ACTIVE/PASSIVENET", performs all the required symbol manipulation in parallel, and handles entire embedded sub-trees (e.g., complex NPs) as readily as it does simple symbols.

ACTIVE/PASSIVENET processes sentences with two possible syntactic structures: simple active sentences of the form  and passive sentences of the form  . Each is transformed into a tree representing V(A, P), namely  . Here, the agent A and patient P of the verb V are both arbitrarily complex noun phrase trees. (The network could actually handle arbitrarily complex V's as well. Aux is taken as a marker of passive, e.g., *are* in *are admired.*)

Figure 1 shows the network processing a passive sentence ((A.B).((Aux.V).(by.C))) as in *Few connectionists are admired by Jerry* and generating (V.(C.(A.B))) as output. The network is presented with an activation vector representing the sentence at the input units, shown at the bottom. This input is fed to the two units labeled "passive" and "active", which decide whether the sentence is a passive or an active one. In the figure, the input sentence is passive and the "passive" unit is turned on. This unit

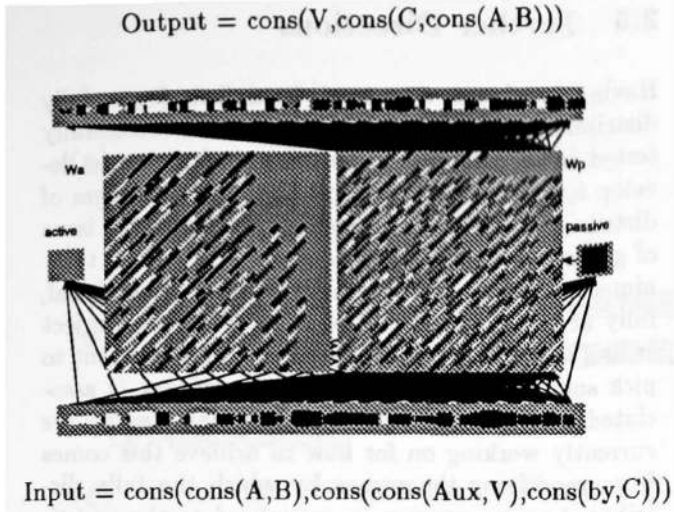

Figure 1: Active/PassiveNET processing a passive sentence

then gates the input pattern through the weight matrix "Wp" to generate an output activation pattern, shown at the top, representing the interpretation of the input sentence. If the input is an active sentence, the "active" unit is turned on, which gates the input pattern through the weight matrix "Wa" instead.

2 A Fully Distributed Recursive Representation

While the tensor product technique is general enough to apply to virtually any kind of symbolic structure, we will consider in this paper only the special case of binary trees, the basic data structure of LISP. The work reported in this section extends earlier results presented in [Legendre et al., 1991, Smolensky, 1990].

2.1 Representation of Binary Trees

A binary tree may be viewed as having a large number of positions with various locations relative to the root: we can adopt *positional roles* r_x labelled by binary strings (or bit vectors) such as $x = 0101$ which is the position in a tree accessed by the LISP function `cadadr` .[3] Decomposing the tree using these structural roles (positions), each constituent of a tree is an atom (the filler) bound to some role r_x specifying its location. A tree $\mathbf{s}$ with a set of atoms $\{\mathbf{f}_i\}$ at respective locations $\{x_i\}$ has the tensor product representation $\mathbf{s} = \sum_i \mathbf{f}_i \otimes \mathbf{r}_{\mathbf{x}_i}$.

The role vectors themselves are also defined as tensor products of sequences of smaller vectors. For example, the role vector for the left child of the right child of the root is:

$$\mathbf{r_{01}} \equiv \mathbf{v} \otimes \mathbf{v} \otimes .... \mathbf{v} \otimes \mathbf{r_0} \otimes \mathbf{r_1}$$

The rightmost vector ($\mathbf{r_1}$ in this case) represents which child of the root (left or right subtree) it belongs to. The second vector from the right represents which (left or right) branch of this subtree it belongs to. More generally stated, each role vector is constructed as the tensor product of a sequence of D vectors; each of these vectors is chosen from a set of three basic vectors $\{\mathbf{r_0}, \mathbf{r_1}, \mathbf{v}\}$. This sequence specifies the position in the tree represented by each role vector. If the position is at depth d in the tree, then the nth vector from the right is: for $n = 1$ through d, $\mathbf{r_0}$ or $\mathbf{r_1}$ depending on whether it is in the right or left branch at depth n; and $\mathbf{v}$ for $n > d$. Basically the vector $\mathbf{v}$ acts like a place holder (like the digit 0) to make the total rank of the tensor product constant (D). As long as the vectors $\{\mathbf{r_0}, \mathbf{r_1}, \mathbf{v}\}$ are linearly independent, trees up to depth D can be represented with complete accuracy. If these vectors are chosen so that each has non-zero components along all coordinate axes, then every unit in the connectionist network will take part in the representation of every atom, regardless of its depth in the tree. We assume henceforth, for simplicity, that these vectors form an orthonormal basis — they are mutually orthogonal and have norms of 1.

2.2 Processing of Binary Trees

This section describes how trees represented using this distributed recursive tensor representation are manipulated. First, take the *cons* operation illustrated in Figure 2 which merges two trees into one. For example, at the symbolic level (Figure-2-(a)), two trees each consisting only of an atom (A and B, respectively) at the root, are merged into a new tree with A and B as the left and the right children of the root. This symbolic operation *cons* is mapped to a connectionist operation (Figure 2-(b) and (c)) that takes two tensor products $\mathbf{T}_A = \mathbf{A} \otimes \mathbf{v} \otimes \mathbf{v} ... \otimes \mathbf{v}$ representing the tree A and $\mathbf{T}_B = \mathbf{B} \otimes \mathbf{v} \otimes \mathbf{v} ... \otimes \mathbf{v}$ representing the tree B, and generates a tensor product that is the sum of the tensor $\mathbf{A} \otimes \mathbf{v} \otimes \mathbf{v} ... \otimes \mathbf{r_0}$ representing A at the left child of the root and the tensor $\mathbf{B} \otimes \mathbf{v} \otimes \mathbf{v} ... \otimes \mathbf{r_1}$ representing B at the right child of the root.

[3] `cadadr(s)` $\equiv$ `car(cdr(car(cdr(s))))` , that is, the left child (0; `car`) of the right child (1; `cdr`) of the left child of the right child of the root of the tree `s` .

to $\mathbf{cons}(\mathbf{cdadr}(\mathbf{s}), \mathbf{cons}(\mathbf{cdddr}(\mathbf{s}), \mathbf{car}(\mathbf{s})))$ by the matrix $\mathbf{W_p} = \mathbf{W_{cons0}W_{cdr}W_{car}W_{cdr}} + \mathbf{W_{cons1}}(\mathbf{W_{cons0}W_{cdr}W_{cdr}W_{cdr}} + \mathbf{W_{cons1}W_{car}})$. In the terminology of production systems, this "action matrix" $\mathbf{W_p}$ is gated by a "condition unit" which determines whether the input pattern $\mathbf{s}$ represents the parse tree $\mathbf{s}$ of a passive sentence, by checking whether $\mathbf{caddr}(\mathbf{s}) = \mathbf{Aux}$. This condition unit can be a linear threshold element whose activity is 1 when its net input $I \geq 0$, and 0 otherwise; its net input $I = -\|\mathbf{W_{car}W_{cdr}W_{cdr}s} - \mathbf{Aux}\|^2$ is always negative, except that $I = 0$ when the desired condition $\mathbf{caddr}(\mathbf{s}) = \mathbf{Aux}$ is satisfied.

2.4 TPPL

We have begun to develop a symbolic formalism—TPPL for "Tensor Product Programming Language"—which enables high-level formal characterization of the computations performed by connectionist networks processing tensor product representations. TPPL contains analogues of simple programming language control structures (like `if-then-else`) and basic symbolic computation operators (like `car`, `cdr`, `cons`) which are formally defined using elements of the tensor calculus. TPPL enables both a formal higher-level symbolic description of the lower-level networks and a calculus for proving their correctness.

To illustrate the idea, ACTIVE/PASSIVENET is described in TPPL by the program:

$$
\begin{aligned}
AP(s) &\equiv \mathtt{if}\ PassiveP(s)\ \mathtt{then}\ PassiveF(s) \\
&\quad\ \mathtt{else}\ ActiveF(s) \\
PassiveP(s) &\equiv [\mathtt{cadr}(s) = \mathtt{Aux}] \\
PassiveF(s) &\equiv \mathtt{cons}(\mathtt{cdadr}(s))), \\
&\quad\ \mathtt{cons}(\mathtt{cdddr}(s))), \mathtt{car}(s))) \\
ActiveF(s) &\equiv \mathtt{cons}(\mathtt{cadr}(s)), \\
&\quad\ \mathtt{cons}(\mathtt{car}(s), \mathtt{cddr}(s))))
\end{aligned}
$$

The primitive operations `car`, `cdr`, `cons` are defined using the inner- and outer-product operations of tensor calculus, as explained above. As illustrated in Sections 1.2 and 2.3, the matrix realizations of these operations are straightforwardly combined to produce a single matrix that performs the entire function *PassiveF*, which can then be implemented in one layer of connection weights. The same applies to the other functions, *ActiveF* and *PassiveMarkerF* which are described in detail in [Smolensky et al., 1992].

2.5 Further Directions

Having produced a general formalism for a fully distributed recursive representation and successfully tested it by computer simulation, it remains to develop specific ways to exploit the full advantages of distributed representation. First, consider the issue of graceful saturation with depth. One general technique is to take a high (perhaps infinite) dimensional, fully accurate representational space and to project onto a lower dimensional subspace. Here we want to pick such a subspace so that less accuracy is associated with greater depth. A promising idea we are currently working on for how to achieve this comes from modifying the means by which the fully distributed representation is specialized to the 'stratified' representation of our earlier work, where different depths of the tree were distributed over different pools of units. Stratification is the special case when $\mathbf{v} = (0, 1)$ because $\mathbf{v}$ is orthogonal to the subspace spanned by $\{\mathbf{r_0}, \mathbf{r_1}\}$. If instead we choose $\mathbf{v}$ to have small but non-zero projection onto this subspace, each depth will have small but non-zero representation on subspaces that are primarily dedicated to the representation of lesser depths. We can therefore set up a fully distributed representation with some large depth limit D (possibly infinite), and then project onto a lower-dimensional subspace, achieving a "soft depth limit" beyond which the representation saturates gracefully.

A second direction for future work is to explore the underlying connectionist distributed represenetations. A central connectionist principle asserts that distributed representations encode the feature- or similarity-structure of information, and a main motivation for distributed tensor product representations is to allow the application of this principle to the *roles* of symbolic structures. Here, natural languages provide a distinctly better domain of study than purely formal languages, because of all the meaningful information that is encoded through real linguistic structure. In a purely formal binary tree, the two recursive roles *left child* and *right child* are just two primitive, distinct roles; in our simulations we represent them as two arbitrary distributed vectors. But linguistic theories provide a rich set of features (explicitly or implicitly) for describing the roles in syntactic and semantic structure. These include hierarchically structured grammatical functions (subject, direct object, ...), thematic roles (agent, patient, ...), X-bar syntactic configurational roles (head, specifier, argument, adjunct ...), syntactic and semantic feature structural roles (number, gender, ...), and many more. Tensor product representations make it possible to study the

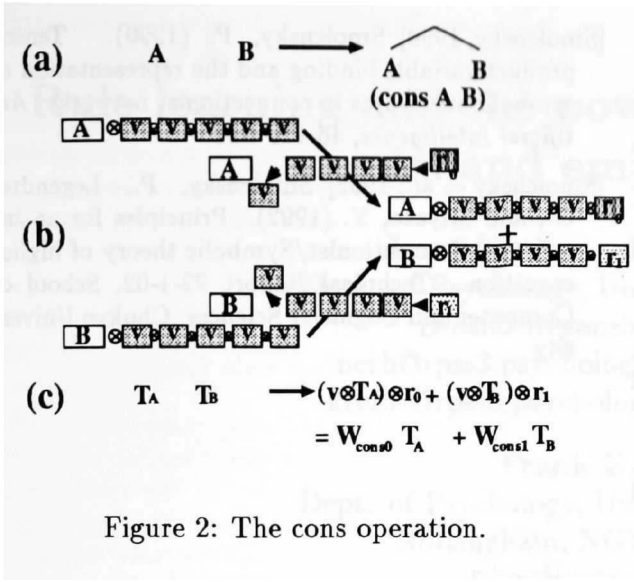

Figure 2: The cons operation.

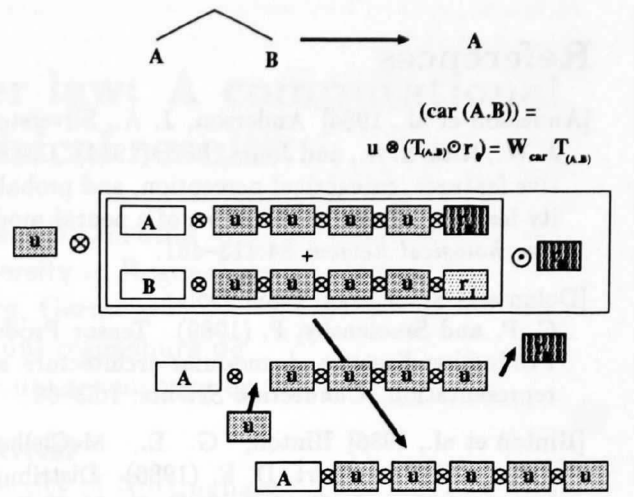

Figure 3: The car operation.

This operation is easily understood as one combining two linear operations that take two trees and push them into the left and the right subtrees of a new tree (let us call these operations $cons_0$ and $cons_1$).

$$cons(A, B) \equiv cons_0(A) + cons_1(B)$$

Each of these operations is achieved, in turn, by combining two basic linear operations, the tensor contraction $\odot$ and the tensor product operation $\otimes$:

$$cons_0(\mathbf{T}) \equiv \mathbf{T} \star \mathbf{r_0} \equiv (\mathbf{v} \odot \mathbf{T}) \otimes \mathbf{r_0}$$
$$cons_1(\mathbf{T}) \equiv \mathbf{T} \star \mathbf{r_1} \equiv (\mathbf{v} \odot \mathbf{T}) \otimes \mathbf{r_1}$$

The tensor contraction operation $\mathbf{v} \odot \mathbf{T}$ replaces the left-most ("deepest") role-vector in $\mathbf{T}$ with its inner product with $\mathbf{v}$. Since the inner product is a scalar, the resulting tensor has rank one less than that of $\mathbf{T}$. $\mathbf{T} \star \mathbf{w}$ is a rank-preserving outer product, a version of $\mathbf{T} \otimes \mathbf{w}$ in which the tensor product with $\mathbf{w}$ is taken only after "unpadding" $\mathbf{T}$ via an inner product with $\mathbf{v}$. In the expression $(\mathbf{v} \odot \mathbf{T}) \otimes \mathbf{r_0}$, the $\odot$ operation takes the leftmost $\mathbf{v}$ vector off the tensor product and the $\otimes$ operation pushes the vector $\mathbf{r_0}$ or $\mathbf{r_1}$ into the tensor product from the right.

Now we consider taking the car or cdr of a tree—extracting its left or right subtree. Defining the car and cdr operations requires the following generalization of the inner product operation.

$$car(\mathbf{T}) \equiv \mathbf{T} \circ \mathbf{r_0} \equiv \mathbf{v} \otimes (\mathbf{T} \odot \mathbf{r_0})$$
$$cdr(\mathbf{T}) \equiv \mathbf{T} \circ \mathbf{r_1} \equiv \mathbf{v} \otimes (\mathbf{T} \odot \mathbf{r_1})$$

$\mathbf{T} \circ \mathbf{w}$ is a rank-preserving inner product, a version of $\mathbf{T} \odot \mathbf{w}$ in which a right inner product with $\mathbf{w}$ is taken, and an extra $\mathbf{v}$ is added to pad the new tensor back up to full rank. As illustrated in Figure 3, the rightmost role-vector is $\mathbf{r_0}$ for the left subtree and $\mathbf{r_1}$ for the right subtree, so the inner product with $\mathbf{r_0}$, used in the *car* operation, is 1 for the left subtree and 0 for the right. This is reversed for the *cdr* operation where inner product is taken with $\mathbf{r_1}$.

2.3 Parallel Implementation

If we regard a tensor representing a tree as a *single* vector in a vector space V, the $cons_0, cons_1, car, cdr$ operations can each be defined as a single (square) matrix that maps a vector in V to another vector in V. Thus, they can be implemented as simple vector-matrix multiplication operations. The four matrices are called $\mathbf{W_{cons0}}$, $\mathbf{W_{cons1}}$, $\mathbf{W_{car}}$, and $\mathbf{W_{cdr}}$.

This connectionist representation/processing of trees enables massively parallel processing. Whereas in the traditional sequential implementation of LISP, symbol processing consists of a long sequence of car, cdr, and cons operations, here we can compose together the corresponding sequence of $\mathbf{W_{car}}$, $\mathbf{W_{cdr}}$, $\mathbf{W_{cons0}}$ and $\mathbf{W_{cons1}}$ operations into a single matrix operation. Adding some minimal nonlinearity allows us to compose more complex operations incorporating the equivalent of conditional branching. For example, a passive sentence parse tree $\mathbf{s}$ is transformed by ACTIVE/PASSIVENET

consequences—for representation, processing, learning, and grammatical description—of directly encoding such information via distributed role vectors.

References

[Anderson et al., 1984] Anderson, J. A., Silverstein, J. W., Ritz, S. A., and Jones, R. S. (1984). Distinctive features, categorical perception, and probability learning: Some applications of a neural model. *Psychological Review*, 84:413–451.

[Dolan and Smolensky, 1989] Dolan, C. P. and Smolensky, P. (1989). Tensor Product Production System: A modular architecture and representation. *Connection Science*, 1:53–68.

[Hinton et al., 1986] Hinton, G. E., McClelland, J. L., and Rumelhart, D. E. (1986). Distributed representation. In Rumelhart, D. E., McClelland, J. L., and the PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, chapter 3, pages 77–109. MIT Press/Bradford Books, Cambridge, MA.

[Legendre et al., 1991] Legendre, G., Miyata, Y., and Smolensky, P. (1991). Distributed recursive structure processing. In Touretzky, D. S. and Lippman, R., editors, *Advances in Neural Information Processing Systems 3*, pages 591–597, San Mateo, CA. Morgan Kaufmann. Slightly expanded version in Brian Mayoh, editor, *Scandinavian Conference on Artificial Intelligence—91*, pages 47–53. IOS Press, Amsterdam.

[McClelland and Kawamoto, 1986] McClelland, J. L. and Kawamoto, A. H. (1986). Mechanisms of sentence processing: Assigning roles to constituents. In Rumelhart, D. E., McClelland, J. L., and the PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 2: Psychological and Biological Models*, chapter 19, pages 272–325. MIT Press/Bradford Books, Cambridge, MA.

[Plate, 1991] Plate, T. (1991). Holographic reduced representations. Technical Report CRG-TR-91-1, Department of Computer Science, University of Toronto.

[Smolensky, 1987b] Smolensky, P. (1987b). On variable binding and the representation of symbolic structures in connectionist systems. Technical report, Department of Computer Science, University of Colorado at Boulder. Technical Report CU-CS-355-87.

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

[Smolensky, 1990] Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist networks. *Artificial Intelligence*, 46:159–216.

[Smolensky et al., 1992] Smolensky, P., Legendre, G., and Miyata, Y. (1992). Principles for an integrated Connectionist/Symbolic theory of higher cognition. Technical Report 92-1-02, School of Computer and Cognitive Sciences, Chukyo University