# Tensor Product Variable Binding and the Representation of Symbolic Structures in Connectionist Systems

**Paul Smolensky**
*Department of Computer Science and*
*Institute of Cognitive Science, University of Colorado,*
*Boulder, CO 80309-0430, USA*

ABSTRACT

*A general method, the tensor product representation, is defined for the connectionist representation of value/variable bindings. The technique is a formalization of the idea that a set of value/variable pairs can be represented by accumulating activity in a collection of units each of which computes the product of a feature of a variable and a feature of its value. The method allows the fully distributed representation of bindings and symbolic structures. Fully and partially localized special cases of the tensor product representation reduce to existing cases of connectionist representations of structured data. The representation rests on a principled analysis of structure; it saturates gracefully as larger structures are represented; it permits recursive construction of complex representations from simpler ones; it respects the independence of the capacities to generate and maintain multiple bindings in parallel; it extends naturally to continuous structures and continuous representational patterns; it permits values to also serve as variables; and it enables analysis of the interference of symbolic structures stored in associative memories. It has also served as the basis for working connectionist models of high-level cognitive tasks.*

## 1. Introduction

Connectionist models rely on parallel numerical computation rather than the serial symbolic computation of traditional AI models, and with the inroads of connectionism has come considerable debate about the roles these two forms of computation should play in AI. While some presume the approaches to be diametrically opposed, and argue that one or the other should be abandoned, others argue that the two approaches are so compatible that in fact connectionist models should just be viewed as implementations of symbolic systems.

In [41] (and also in [36, 38]) I have argued at considerable length for a more complex view of the roles of connectionist and symbolic computation in cognitive science. A one-sentence summary of the implications of this view for AI is this: connectionist models may well offer an opportunity to escape the

*Artificial Intelligence* **46** (1990) 159–216
0004-3702/90/$03.50 

brittleness of symbolic AI systems, a chance to develop more human-like intelligent systems—but only if we can find ways of naturally instantiating the sources of power of symbolic computation within fully connectionist systems. If we ignore the connectionist approach, we may miss an excellent opportunity for formally capturing the subtlety, robustness, and flexibility of human cognition, and for elucidating the neural underpinnings of intelligence. If we ignore the symbolic approach, we throw out tremendous insights into the nature of the problems that must be solved in creating intelligent systems, and of techniques for solving these problems; we probably doom the connectionist approach to forever grappling with simple cognitive tasks that fall far short of the true capacity of human intelligence. If we use connectionist systems merely to implement symbolic systems, we might get AI systems that are faster and more tolerant of hardware faults, but they will be just as brittle.

The present paper is part of an effort to extend the connectionist framework to naturally incorporate the ingredients essential to the power of symbolic computation, without losing the virtues of connectionist computation. This extended version of connectionist computation would integrate, in an intimate collaboration, the discrete mathematics of symbolic computation and the continuous mathematics of connectionist computation. This paper offers an example of what such a collaboration might look like.

One domain where connectionist computation has much to gain by incorporating some of the power of symbolic computation is language. The problems here are extremely fundamental. Natural connectionist representation of a structured object like a phrase-structure tree—or even a simple sequence of words or phonemes—poses serious conceptual difficulties, as I will shortly discuss. The problem can be traced back to difficulties with the elementary operation of binding a value to a variable.

I begin in Section 1.1 by discussing why natural connectionist representation of structured objects is a problem. I list several properties of the solution to this problem that is presented in this paper. In Section 1.2, I respond to the possible connectionist criticism that it is misguided to even try to solve this problem. Then, in Section 1.3, I outline the rest of the paper.

Before proceeding it is worth commenting on where the research reported here fits into an overall scheme of connectionist AI. As in the traditional approach, in the connectionist approach several components must be put together in constructing a model. Elements of the task domain must be represented, a network architecture must be designed, and a processing algorithm must be specified. If the knowledge in the model is to be provided by the designer, a set of connections must be designed to perform the task. If the model is to acquire its knowledge through learning, a learning algorithm for adapting the connections must be specified, and a training set must be designed (e.g., a set of input/output pairs). For most of these aspects of connectionist modeling, there exists considerable formal literature analyzing the problem and

offering solutions. There is one glaring exception: the representation component. This is a crucial component, for a poor representation will often doom the model to failure, and an excessively generous representation may essentially solve the problem in advance. Representation is particularly critical to understanding the relation between connectionist and symbolic computation, for the representation often embodies most of the relation between a symbolically characterized problem (e.g. a linguistic task) and a connectionist solution.

Not only is the connectionist representation problem a central one, it is also a problem that is amenable to formal analysis. In this paper the problem will be characterized as finding a mapping from a set of structured objects (e.g. trees) to a vector space, the set of states of the part of a connectionist network representing those objects. The mélange of discrete and continuous mathematics that results is reminiscent of a related classical area of mathematics: the problem of representing abstract groups as collections of linear operators on a vector space. The discrete aspects of group theory and the continuous aspects of vector space theory interact in a most constructive way. Group representation theory, with its application to quantum physics, in fact offers a useful analogy for the connectionist representation of symbolic structures. The world of elementary particles involves a discrete set of particle species whose properties exhibit many symmetries, both exact and approximate, that are described by group theory. Yet the underlying elementary particle state spaces are continuous vector spaces, in which the discrete structure is imbedded. In the view that guides the research reported here, in human language processing, the discrete symbolic structures that describe linguistic objects are actually imbedded in a continuous connectionist system that operates on them with flexible, robust processes that can only be approximated by discrete symbol manipulations.

One final note on terminology. In most of this paper the structures being represented will be referred to as *symbolic structures*, because the principal cases of interest will be objects like strings and trees. Except for the consideration of particular symbolic structures, however, the analysis presented here is of structured objects in general; it therefore applies equally well to objects like images and speech trains which are not typically considered "symbolic structures." With this understood, in general discussions I will indiscriminately refer to objects being represented as "structures," "structured objects," or "symbolic structures."

### 1.1. Distributed representation and variable binding in connectionist systems

I have called the problem considered in this paper that of finding "natural" connectionist representation of structured objects and variable bindings. In fact what I refer to is the problem of finding connectionist representations that are *fully distributed*. The notion of *distributed representation* is central to the power

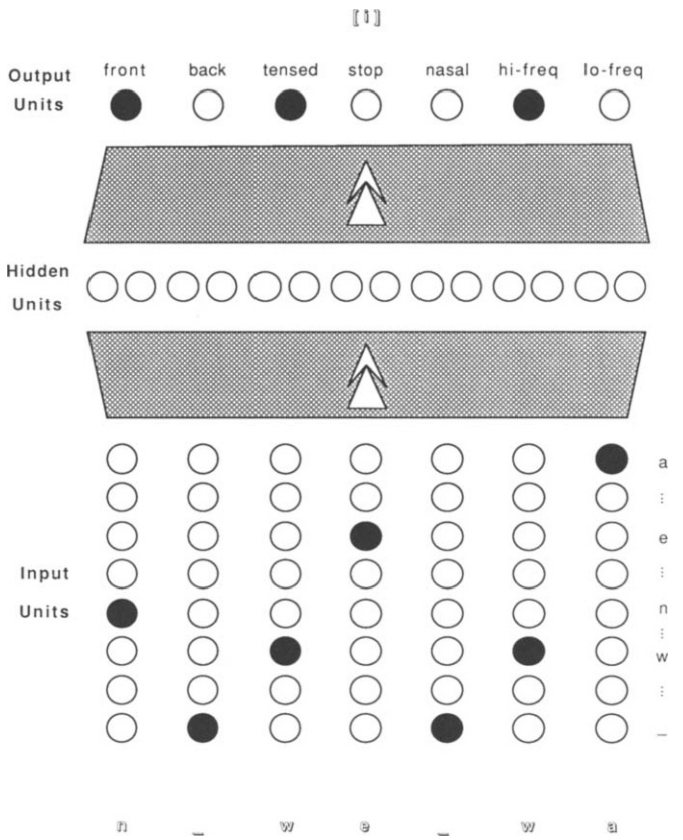


Fig. 1. The NETtalk system of Sejnowski and Rosenberg [35] illustrates both distributed and local connectionist representations.

of the connectionist approach (e.g., [1, 15, 24, 34, 37, 41]; for an opposing view, see [10]). To illustrate the idea of distributed representation, consider the NETtalk system, a connectionist network that learns to pronounce written English [35] (see Fig. 1). Each output of this network is a phonetic segment, e.g. the vowel [*i*] in the pronunciation of the word *we*. Each phonetic segment is represented in terms of phonetic features; for [*i*], we have: front = 1, tensed = 1, high-frequency = 1, back = 0, stop = 0, nasal = 0, and so forth. There is one output processor in the network for each of the phonetic features, and its numerical value indicates whether that feature is present (1) or absent (0). Each phonetic segment is therefore represented by a *pattern of activity* over the numerous output processors, and each output processor participates in the representation of many different outputs. This defines a distributed representation of the output phonetic segment.

At the opposite end of a connectionist representational spectrum are *local representations*. These too are illustrated by NETtalk; this time, in the input. Each NETtalk input consists of a letter to be pronounced together with the three preceding and three following letters to provide some context. For each of these seven letters there is a separate pool of input processors in the network, and within each pool there is a separate processor for each letter. In the input representation, in each of the seven pools the single processor corresponding to the letter present is assigned activity 1, and the remaining processors in the pool are all assigned activity 0. This representation is local in two senses. Most obviously, different letters are represented by activity in disjoint localities—in single processing units. Unlike the output activity, there is no overlap of the activity representing alternative values. The other sense of locality is that the activity representing different letter *positions* are all disjoint: each pool localizes the activity representing one letter position.

The input representation in NETtalk illustrates the problems of representing structures and of variable binding in connectionist networks. The input is a string of seven characters (of which the middle one is to be pronounced). There is a pool of processing units dedicated to representing each item in this string. Each pool can be viewed as a slot in the structure: a variable. The value of each variable is a pattern of activity residing in its pool of units. In NETtalk this pattern is localized; we will later consider examples of models in which the corresponding pattern is distributed throughout the pool. Regardless of the patterns used to represent the values, in such systems *the variables are localized regions of the network*. These variables are in fact *registers*, and the great majority of connectionist systems have represented structured data using them. Yet registers are hardly natural or desirable within connectionist models. In order to make available in the processing of structured data the full power of connectionist computation that derives from distributed representation, we need to use *distributed representations of variables* in addition to distributed representations of values.

In this paper a completely distributed representational scheme for variable binding is analyzed: the *tensor product representation*. The tensor product of an $n$-dimensional vector $\mathbf{v}$ and an $m$-dimensional vector $\mathbf{w}$ is simply the $nm$-dimensional vector $\mathbf{v} \otimes \mathbf{w}$ whose elements are all possible products $v_i w_j$ of an element of $\mathbf{v}$ and an element of $\mathbf{w}$. This vector $\mathbf{v} \otimes \mathbf{w}$ is a *tensor of rank two*: its elements are labelled by two indices $i$ and $j$. A tensor of rank one is simply an ordinary vector labelled by a single index, and a tensor of rank zero is a scalar. Tensors of rank higher than two arise by taking tensor products of more than two ordinary vectors; if $\mathbf{w}$ is an $l$-dimensional vector, then $\mathbf{u} \otimes \mathbf{v} \otimes \mathbf{w}$ is a tensor of rank three, the $nml$-dimensional vector consisting of all products $u_i v_j w_k$. The tensor product generalizes the matrix algebra concept of outer product to allow third- and higher-order products; the more general apparatus of tensor

algebra is needed here because the recursive use of tensor product representations leads to tensors of rank higher than two.[1]

In the tensor product representation, both the variables and the values can be arbitrarily nonlocal, enabling (but not requiring) representations in which every unit is part of the representation of every constituent in the structure. In this paper, applications of the tensor product scheme to the connectionist representation of complex structured objects are explored. Features of the tensor product representation, most of which distinguish it from existing representations, include the following (corresponding section numbers are indicated in parentheses):

- The representation rests on a principled and general analysis of structure: role decomposition (Section 2.2.1).
- A fully distributed representation of a structured object is built from distributed representations of both the structure's constituents and the structure's roles (Section 2.2.4).
- Nearly all previous connectionist representations of structured data, employing varying degrees of localization, are special cases (Section 2.3).
- If a structure does not saturate the capacity of a connectionist network that represents it, the components of the structure can be extracted with complete accuracy (Section 3.1).
- Structures of unbounded size can be represented in a fixed connectionist network, and the representation will saturate gracefully (Section 3.2).
- The representation applies to continuous structures and to infinite networks as naturally as to the discrete and finite cases (Section 3.3).
- The binding operations can be simply performed in a connectionist network (Section 3.4).
- The representation respects the independence of two aspects of parallelism in variable binding: generating vs. maintaining bindings (Section 4.1).
- The constituents of structure can be simply extracted in a connectionist network (Section 3.4.2).
- A value bound to one variable can itself be used as a variable (Section 3.6).
- The representation can be used recursively, and connectionist representations of operations on symbolic structures, and recursive data types, can be naturally analyzed (Section 3.7).
- Retrieval of representations of structured data stored in connectionist memories can be formally analyzed (Section 3.8).

[1] For treatments of tensor algebra, see [19, 26, 48]; for a short presentation directed to the current work, see [39, Appendix]. While this paper will not make extensive use of tensor calculus, setting the connectionist issues discussed here in the framework of tensor algebra provides a useful link to well-established mathematics. Certain virtues of the tensor calculus, such as the way it systematically manages the multiple indices associated with higher-rank tensors, have already proved important for actual connectionist systems built on tensor product representations [3–7].

- A general sense of optimality for activity patterns representing roles in structures can be defined and analyzed (Section 3.9).
- A connectionist "recirculation" learning algorithm can be derived for finding these optimal representations (Section 3.9).

### 1.2. Connectionist representation of symbolic structures

The general issue behind the research reported here is the representation in connectionist systems of symbolic structures. This issue figures prominently in the argument of Fodor and Pylyshyn [11] that connectionist models are inadequate on fundamental representational grounds, and the work reported here began as a response to this attack; for responses to Fodor and Pylyshyn based in part on the work reported here, see [40, 42].

A more concrete motivation for pursuing this issue are the challenges facing connectionist modeling of higher cognitive processes such as language. Here our central question takes the form: What are computationally adequate connectionist representations of strings, trees, sentences?

This section is addressed to connectionists who may find this question misguided. The essence of the connectionist approach, they might say, is to expunge symbolic structures from models of the mind. I must agree that the connectionist approach is rather far from a "language of thought" view of cognition in which all mental states are formalized as symbolic structures. However there still remains in connectionism an important role to be played by language and symbolic structures, even if that role is substantially reduced relative to its counterpart in the traditional radically symbolic approach. I have argued this point in some detail in [41], and will only summarize the relevant conclusions here.

Any connectionist model of natural language processing must cope with the questions of how linguistic structures are represented in connectionist models. A reasonable starting point would seem to be to take linguistic analysis of the structure of linguistic objects seriously, and to find a way of representing this structure in a connectionist system. Since the majority of existing representations of linguistic structure employ structures like trees and strings, it is important to find adequate connectionist representations of these symbolic structures. It may well turn out that once such representations are understood, new connectionist representations of linguistic structures will be developed that are not truly representations of symbolic structures but which are more adequate according to the criteria of linguistics, computational linguistics, psycholinguistics, or neurolinguistics. It seems likely, however, that such improvements will rest on prior understanding of connectionist representations of existing symbolic descriptions of linguistic structure.

The importance to the connectionist approach of representing linguistic structures goes well beyond models of natural language processing. Once

adequate connectionist representations are found for linguistic structures, then these can serve as the basis for connectionist models of conscious, serial, rule-guided behavior. This behavior can be modeled as explicit (connectionist) retrieval and interpretation of linguistically structured rules. Adequate connectionist models of such behavior are important for connectionist models of higher thought processes.

One line of thought in the connectionist approach implies that analyses of connectionist representations of symbolic structures are unnecessary. The argument goes something like this. Just as a child somehow learns to internally represent sentences with no explicit instruction on how to do so, so a connectionist system with the right learning rule will somehow learn the appropriate internal representations. The problem of linguistic representation is not to be solved by a connectionist theorist but rather a connectionist network.

In response to this argument I have five points.

(1) In the short term, at least, our learning rules and network simulators do not seem powerful enough to make network learning of linguistic representation feasible.
(2) Even if such learning is feasible at some future point, we will still need to *explain* how the representation is done. There are two empirical reasons to believe that such explanation will require the kind of analysis begun in this paper: explanation of the computation of real neural networks has turned out to require much analysis, as mere observation has proved woefully inadequate; the same has turned out to be true even of the self-organized connectionist networks that perform computations vastly simpler than most of natural language processing.
(3) It is important to try to build bridges as soon as possible between connectionist accounts of language processing and existing accounts; the problem is just too difficult to start all over again from scratch.
(4) We would like to be able to experiment *now* with connectionist learning models of rather complex linguistic skills (e.g. parsing, anaphoric resolution, and semantic interpretation, all in complex sentences). For now, at least, such experiments require connectionist representation of linguistic structures to serve as inputs and outputs. We want to study the learning of the operations performed on linguistic structures without waiting many years for the completion of the study of the learning of the linguistic representations themselves.
(5) Language is more than just a domain for building models, it is a foundation on which the entire traditional theory of computation rests. To understand the computational implications of connectionism, it is crucial to know how the basic concepts of symbolic computation and formal language theory relate to connectionist computation.

Of course, exploiting connectionist representations of the sort of symbolic structures used in symbolic AI by no means commits one to a full connectionist implementation of symbolic AI, which, as stated earlier, would miss most of the point of the connectionist approach. The semantic processing of a connectionist representation of a parse tree should not be performed by a connectionist implementation of serially applied symbolic rules that manipulate the tree; rather, the processing should be of the usual connectionist sort: massively parallel satisfaction of multiple soft constraints involving the micro-elements forming the distributed representation of the parse tree. Thus in this paper connectionist representations of *pop* and *cdr* will be mathematical relations between patterns of activity, not processes carried out over time in a connectionist network as part of an extended serial computation (in contrast to [44]). The view behind the present research is not that mental operations are always serial symbol manipulations (although a few are); rather the view is that the information processed often has useful symbolic *descriptions*, and that these descriptions should be taken seriously. (This view is spelled out in detail in [41].)

### 1.3. Outline of the paper

In Section 2, the notion of connectionist representation is formally defined and the tensor product representation is constructed. Examples are considered, and the various special cases that reduce to previous connectionist representations are discussed. In Section 3, a number of properties of the tensor product representation are proved and several extensions discussed. The connectionist representation of symbolic operations is defined, and examples for strings and trees are considered. Retrieval of symbolic structures represented in connectionist memories by the tensor product representation is analyzed. Finally, work reported elsewhere is briefly summarized concerning a sense of optimality for patterns representing roles in structures, and a connectionist "recirculation" algorithm for learning these optimal representations. Section 4 is a summary and conclusion.

## 2. Connectionist Representation and Tensor Product Binding: Definition and Examples

In this section I first formally characterize the notion of connectionist representation. Next, the problem of representing structured objects is reduced to three subproblems: decomposing the structures via roles, representing conjunctions, and representing variable/value bindings. First role decompositions are discussed, and then the superpositional representation of conjunction and the tensor product representation for variable/value bindings is defined. Next I show how various special cases of the tensor product representation yield the previous connectionist representations of structured data.

### 2.1. Connectionist representation

In this paper, the question of how to represent symbolic structures in connectionist systems will be formalized as follows.

Connectionist representations are patterns of activity over connectionist networks; these patterns can extend over many processors in the network, as in distributed representations, or be localized to a single processor, as in a local representation. Such a pattern is a collection of activation values: a vector with one numerical component for every network processor. The space of representational states of a connectionist network thus lies in a vector space, the dimension of which is equal to the number of processors in the network. Each processor corresponds to an independent basis vector; this forms a distinguished basis for the space. In many connectionist networks the processor's values are restricted in some way; such restrictions are important for consideration of the dynamics of the network but are not central to the representational issues considered here, and they will be ignored. (For expositions of the application of vector space theory—linear algebra—to connectionist systems, see, e.g., [16, 37].)

**Definition 2.1.** The *activity states of a connectionist network* are the elements of a vector space $V$ which has a distinguished basis $\{\hat{\mathbf{v}}_i\}$.

Whenever I speak of a vector space representing the states of a connectionist network, a distinguished basis will be implicitly assumed. Rarely will it be necessary to deal explicitly with this basis. Sometimes it will be useful to use the canonical inner product associated with the distinguished basis: the one in which the basis vectors are orthogonal and of unit norm. (Equivalently, this inner product of two vectors can be computed as the sum of the products of corresponding vector components with respect to the distinguished basis). Whenever I speak of activity patterns being orthogonal, or of their norm, these concepts are taken to be defined with respect to this canonical inner product; the inner product of vectors $\mathbf{u}$ and $\mathbf{v}$ will be denoted $\mathbf{u} \cdot \mathbf{v}$.

**Definition 2.2.** A *connectionist representation* of the symbolic structures in a set $S$ is a mapping $\Psi$ from $S$ to a vector space $V$.

The kinds of sets $S$ we will study are sets of strings and sets of binary trees. Of central concern are the images under the mapping $\Psi$ of the relations between symbolic structures and their constituents, and the images of the operations transforming symbolic structures into other structures. Also important are basic questions about the representation mapping such as whether distinguishable symbolic structures have distinguishable representations:

**Definition 2.3.** A connectionist representation $\Psi$ is *faithful* iff it is one-to-one and there is no structure that it maps to the zero vector $\mathbf{0} \in V$.

### 2.2. Tensor product representation: Definition

The representation of structured objects explored in this paper requires first that structures be viewed as possesing a number (possibly unbounded) of *roles* which, for particular instances of the structure, are individually bound to particular *fillers*. For example, a string may be viewed as possessing an infinite set of roles $\{r_1, r_2, \ldots\}$ where $r_i$ is the role of the $i$th element in the string. A particular string of length $n$ involves binding the first $n$ roles to particular fillers. For example, the string *aba* involves the bindings $\{a/r_1, b/r_2, a/r_3\}$, using a notation in which $f/r$ denotes the binding of filler $f$ to role $r$; in this string, the roles $r_i$ for $i > 3$ are all unbound. Now note that the structure has been characterized as the *conjunction* of an unordered set of variable bindings. The problem of representing the structure has been reduced to the problems of

(1) representing the structure as a conjunction of filler/role bindings;
(2) representing the conjunction operation;
(3) representing the bindings in a connectionist network.

These problems are respectively considered in Sections 2.2.1 through 2.2.3 and brought together in Section 2.2.4.

In Section 3.7.3, we will see that the representations we build using roles and fillers are equivalent to those we would have built by viewing structures as a number of elements engaged in certain structural relations.

#### 2.2.1. *Role decompositions of symbolic structures*

As a formal definition of roles and fillers, I will take the following:

**Definition 2.4.** Let $S$ be a set of symbolic structures. A *role decomposition* $F/R$ for $S$ is a pair of sets $(F, R)$, the sets of *fillers* and *roles*, respectively, and a mapping

$$\mu_{F/R} : F \times R \to Pred(S) ; \qquad (f, r) \mapsto f/r .$$

For any pair $f \in F$, $r \in R$, the predicate on $S$ $\mu_{F/R}(f, r) = f/r$ is expressed: $f$ *fills role* $r$.

The role decomposition has *single-valued roles* iff for any $s \in S$ and $r \in R$, there is at most one $f \in F$ such that $f/r(s)$ holds.

The role decomposition is *recursive* iff $F = S$.

A role decomposition determines a mapping

$$\beta : S \to 2^{F \times R} ; \qquad s \mapsto \{(f, r) \mid f/r(s)\}$$

that associates to each $s \in S$ the set $\beta(s)$ of *filler/role bindings in* $s$. The mapping $\beta$ will be called the *filler/role representation* of $S$ induced by the role decomposition.

The role decomposition is *faithful* iff $\beta$ is one-to-one.

The role decomposition is *finite* iff for each $s \in S$, the set of bindings in $s$, $\beta(s)$, is finite.

Throughout this paper all role decompositions will be assumed to be finite, except in sections where the infinite case is explicitly considered.

Recursive role decompositions are heavily used in the standard description of symbolic structures. For example, the description of a LISP S-expression as a structure whose *car* and *cdr* are both S-expressions is a recursive decomposition via the roles *car* and *cdr*. The tensor product representation to be presented shortly will be used to analyze recursive role decompositions (Section 3.7), but in this paper, the representations we consider will not be *defined* using recursive role decompositions; that is possible, but goes beyond the scope of this paper.

Faithful role decompositions are particularly useful because the filler/role representations they induce allow us to identify each symbolic structure with a predicate having a simple conjunctive form which forms the basis of tensor product representation:

**Theorem 2.5.** *Let $F/R$ be a role decomposition of $S$. For each $s_0 \in S$, define the predicate $\pi_{s_0}$ by:*

$$\pi_{s_0}(s) = \bigwedge_{(f,r)\in\beta(s_0)} f/r(s)$$

*where $\wedge$ denotes conjunction; $\pi_{s_0}(s)$ is true iff the structure $s$ contains all the filler/role bindings in $s_0$. Then if the role decomposition is faithful, the structure $s_0$ can be recovered from the predicate $\pi_{s_0}$.*

**Proof.** This result follows immediately from the following lemma:

**Lemma 2.6.** *The mapping $\beta$ of the role decomposition maps elements of $S$ into subsets of $F \times R$. These subsets possess a partial order, set inclusion $\subseteq$, which can be pulled back to $S$ via $\beta$:*

$$s_1 \preccurlyeq s_2 \quad \textit{iff} \quad \beta(s_1) \subseteq \beta(s_2)\,.$$

*Suppose $F/R$ is faithful. Then with respect to the partial order $\preccurlyeq$, the set of elements of $S$ for which the predicate $\pi_{s_0}$ holds has a unique least element, which is $s_0$. In this way $s_0$ can be recovered from its corresponding predicate $\pi_{s_0}$.*

**Proof of Lemma 2.6.** Since $\beta(s)$ is the set of filler/role bindings in $s$, $s_1 \leqslant s_2$ iff the bindings in $s_1$ are a subset of those of $s_2$:

$$s_1 \leqslant s_2 \quad \text{iff} \quad [\text{for all } f \in F \text{ and } r \in R,\ f/r(s_1) \Rightarrow f/r(s_2)] .$$

Now consider the set of elements $s$ satisfying the predicate $\pi_{s_0}$:

$$\begin{aligned} S(\pi_{s_0}) &:= \{s \in S \mid \pi_{s_0}(s)\} \\ &= \{s \in S \mid \text{for all } f \in F \text{ and } r \in R,\ f/r(s_0) \Rightarrow f/r(s)\} \\ &= \{s \in S \mid s_0 \leqslant s\} . \end{aligned}$$

This set contains $s_0$, and $s_0$ is a least element; it remains to show that there is no other least element. Consider any other element $s_1$ in $S(\pi_{s_0})$. Since $\mu$ is faithful and $s_1 \neq s_0$, there is at least one binding $f/r$ not shared by $s_0$ and $s_1$. Since $s_1 \in S(\pi_{s_0})$ and $s_0$ is a least element of $S(\pi_{s_0})$, we must have $f/r(s_1) \wedge \neg f/r(s_0)$. This implies $\neg(s_1 \leqslant s_0)$ so $s_1$ cannot be a least element in $S(\pi_{s_0})$ □

2.2.2. *Connectionist representation of conjunction*

The representation of conjunction in connectionist models has traditionally been performed with pattern superposition, i.e. vector addition. If two propositions are each represented in a connectionist system by some pattern of activity, the representation of the conjunction of those propositions is the pattern resulting from superimposing the individual patterns. This paper adopts this method. In terms of the representation mapping $\Psi$, we can write:

**Definition 2.7.** A connectionist representation $\Psi$ employs the *superpositional representation of conjunction* iff:

$$\Psi\left(\bigwedge_i p_i\right) = \sum_i \Psi(p_i) .$$

The representation of the conjunction of a collection of propositions is the sum of the representations of the individual propositions.

Note that, like conjunction, vector addition is an operation possessing the properties of associativity and commutativity. Were this not so, vector addition could not be used to represent conjunction.

Applying the superpositional representation of conjunction to the case at hand:

**Definition 2.8.** Suppose $S$ is a set of symbolic structures and $F/R$ is a role decomposition of $S$ with fillers $F$ and roles $R$. Suppose that $\Psi_b$ is a connection-

ist representation of the filler/role bindings:

$$\Psi_b : \{ f/r \mid f \in F, r \in R \} \to V$$

where $V$ is a vector space. Then $\Psi_{F/R}$, the connectionist representation of $S$ induced by $F/R$, the superpositional representation of conjunction, and $\Psi_b$, is

$$\Psi_{F/R} : S \to V ; \qquad s \mapsto \sum_{(f,r) \in \beta(s)} \Psi_b(f/r) .$$

The use of vector addition to represent conjunction has pervasive implications for the faithfulness of representations. If the representations of $a \wedge b$ and $c \wedge d$ are to be distinguishable, then $\mathbf{a} + \mathbf{b}$ and $\mathbf{c} + \mathbf{d}$ must be different. This constrains the possible patterns **a**, **b**, **c** and **d** that can represent $a$, $b$, $c$ and $d$. It will be guaranteed that $\mathbf{a} + \mathbf{b}$ and $\mathbf{c} + \mathbf{d}$ will be different if the vectors **a**, **b**, **c** and **d** are all *linearly independent*: no one can be expressed as a weighted superposition of the others. In order to guarantee the faithfulness of representations, it will often be necessary to impose the restriction of linearly independent representing patterns for the constituents. This restriction is an expensive one, however, since to have $n$ linearly independent patterns one must have at least $n$ nodes in the network. And, as we will see below in Section 3.8, some sets of structures contain so many shared bindings that they cannot be given linearly independent representations, no matter how the bindings are represented; this is true, for example, of the set of strings $\{ax, bx, ay, by\}$, decomposed by the roles of *first_position*, *second_position*.

The addition operation used for superposition in this paper is arithmetic addition, in which $1 + 1 = 2$; in other words, the scalars for the vector spaces are real numbers under numerical addition. An important variation of this analysis would be to consider vector spaces over *Boolean* scalars under logical addition (OR): $1 + 1 = 1$. This variation can still be regarded as tensor product representation, but with respect to a different set of scalars; the result would have quite a different character. Boolean tensor product representations are needed to exactly describe a number of existing connectionist models that use Boolean-valued units and correspondingly use Boolean rather than numerical superposition to represent conjunction. Comparing real-valued and Boolean tensor product representations in general is like comparing connectionist models with real-valued units to those with Boolean units: there are a variety of advantages and disadvantages to each. To mention just two, threshold operations can often allow retrievals from lightly-loaded Boolean systems to be exact [50]; but, on the whole, the mathematics of real scalars is simpler. For this reason, this paper begins the analysis of tensor product representations with the real-valued case.

### 2.2.3. *Connectionist representation of variable binding*

It remains to consider the representation of filler/role bindings; this section introduces the tensor product representation.

The tensor product representation of a value/variable binding is quite simple to define (see Fig. 2). To bind a filler $f$ to a role $r$ we first represent $f$ as a pattern of activity $\mathbf{f}$ over a set of "filler" units $\{\tilde{f}_\phi\}$ and represent $r$ as a pattern of activity $\mathbf{r}$ over a set of "role" units $\{\tilde{r}_\rho\}$. The binding $f/r$ is represented by a pattern of activity $\mathbf{f/r}$ over a set of "binding" units $\{\tilde{b}_{\phi\rho}\}$, of which there is one for each pair of filler and role units. The activity of the binding unit $\tilde{b}_{\phi\rho}$ is the activity of the filler unit $\tilde{f}_\phi$ in the pattern $\mathbf{f}$ times the activity of the role unit $\tilde{r}_\rho$ in the pattern $\mathbf{r}$.

This procedure is readily characterizable in vector terminology. The representation of the role $r$ is a vector $\mathbf{r}$ in a vector space $V_R$. $V_R$ is a real vector space with dimension equal to the number of units $\tilde{r}_\rho$. The representation of the filler $f$ is a vector $\mathbf{f}$ in a vector space $V_F$, a real vector space with dimension equal to the number of units $\tilde{f}_\phi$. The representation of the binding $f/r$ is the *tensor product* vector $\mathbf{f/r} = \mathbf{f} \otimes \mathbf{r}$ in the tensor product vector space $V_B = V_F \otimes V_R$. $V_B$ is a real vector space with dimension equal to the product of the dimensions of $V_F$ and $V_R$. The components of the vector $\mathbf{f/r}$ are related to the components of $\mathbf{f}$ and $\mathbf{r}$ as follows. Each filler unit $\tilde{f}_\phi$ corresponds to a vector $\hat{\mathbf{f}}_\phi$ in $V_F$ (the vector representing the pattern of activity in which that unit has activity 1 and all other units have activity zero). The complete set of vectors

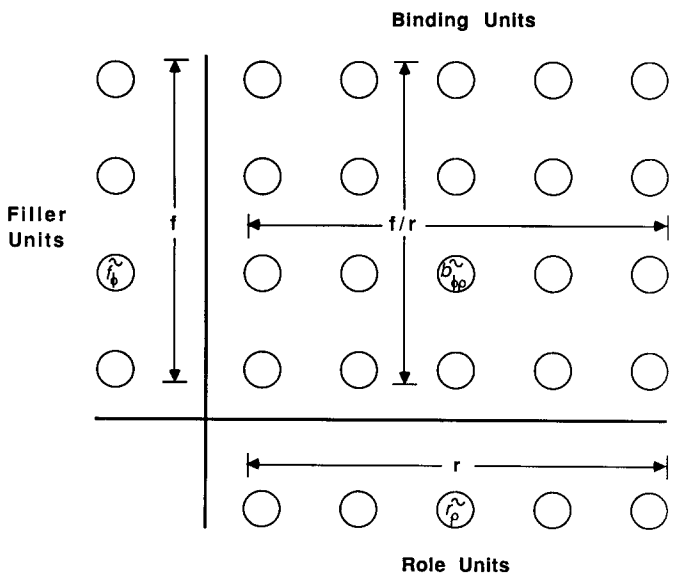


Fig. 2. The tensor product representation for filler/role bindings.

$\{\hat{\mathbf{f}}_\phi\}$ forms the distinguished basis for $V_F$ and any vector $\mathbf{f}$ in $V_F$ can be expressed in terms of this basis as a sequence of real numbers; these are the activities of all the units in the pattern corresponding to $\mathbf{f}$. Exactly the same story holds for the roles. Then the tensor product space $V_B = V_F \otimes V_R$ has as a basis the set of vectors $\{\hat{\mathbf{b}}_{\phi\rho} = \hat{\mathbf{f}}_\phi \otimes \hat{\mathbf{r}}_\rho\}$. The $\phi\rho$ component $(b_{\phi\rho} = (f/r)_{\phi\rho})$ of the vector $\mathbf{b} = \mathbf{f}/\mathbf{r} = \mathbf{f} \otimes \mathbf{r}$ representing the binding is the product of the $\phi$ component of $\mathbf{f}$ $(f_\phi)$ and the $\rho$ component of $\mathbf{r}$ $(r_\rho)$:

$$b_{\phi\rho} = f/r_{\phi\rho} = f_\phi r_\rho \, .$$

**Definition 2.9.** Let $F/R$ be a role decomposition of $S$. Let $\Psi_F$ and $\Psi_R$ be connectionist representations of the fillers and roles:

$$\Psi_F : F \to V_F \, , \qquad \Psi_R : R \to V_R \, .$$

Then the *tensor product representation of the filler/role bindings* induced by $\Psi_F$ and $\Psi_R$ is the mapping:

$$\Psi_b : \{f/r \mid f \in F, r \in R\} \to V_F \otimes V_R \, ; \qquad f/r \mapsto \Psi_F(f) \otimes \Psi_R(r) \, .$$

Fig. 3 shows an example specially chosen for visual transparency. Consider an application to speech processing, and imagine that we are representing the

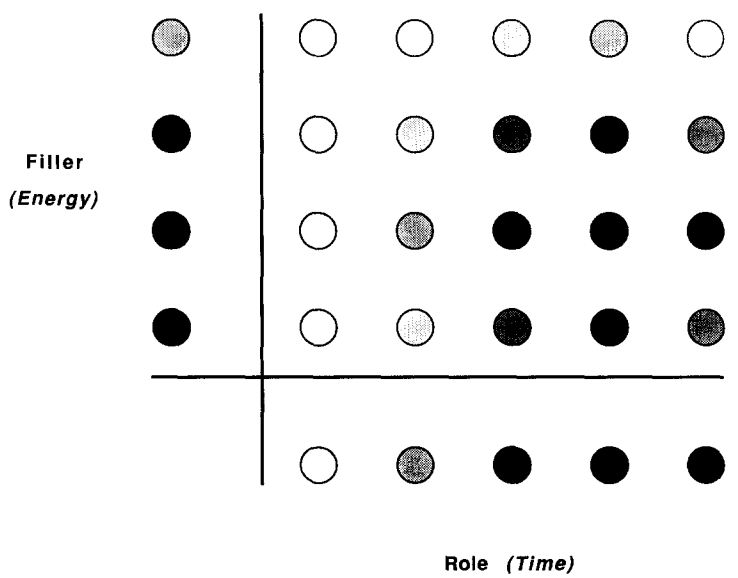


Fig. 3. A visually transparent example of the tensor product representation of a filler/role binding. Darker units are more active.

amount of energy in a particular frequency band over time. For the roles here we take a series of time points and for the fillers the amount of energy in the band. In Fig. 3, the roles are represented as patterns of activity over five units. Each role $r_\rho$ is a time point and is represented as a peaked pattern centered at unit $\rho$; the figure shows the case $\rho = 4$. Each filler $f_\phi$ is an energy level; in Fig. 3 this is represented as a pattern of activity over four units: a single peak centered at the energy level being represented. The binding pattern is a two-dimensional peak centered at the point whose $x$- and $y$-coordinates are the time and energy values being bound together.

The example of Fig. 3 is visually transparent because of the simple geometrical structure of the patterns. Of course there is nothing in the binding mechanism itself that requires this. The distributed representation of roles and fillers can be arbitrary patterns and in general the tensor product of these patterns will be even more visually opaque than are the patterns for the roles and fillers: see Fig. 4. However the mathematical simplicity of tensor product binding makes the general case as easy to analyze as special cases like that of Fig. 3.

2.2.4. *Tensor product representation*

Putting together the previous representations, we have:

**Definition 2.10.** Let $F/R$ be a role decomposition of $S$, and let $\Psi_F$ and $\Psi_R$ be connectionist representations of the fillers and roles. Then the corresponding

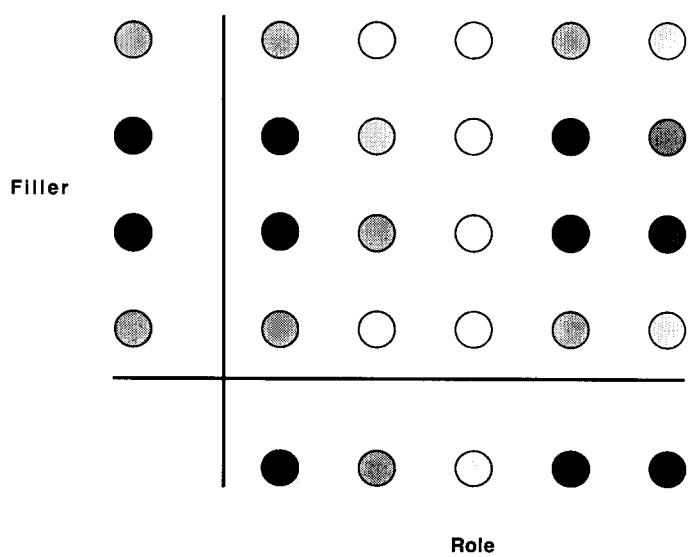


Fig. 4. A generic example of the tensor product representation of a filler/role binding.

*tensor product representation* of $S$ is

$$\Psi : S \to V_F \otimes V_R ; \qquad s \mapsto \sum_{(f,\, r) \in \beta(s)} \Psi_F(f) \otimes \Psi_R(r) .$$

If we identify $s$ with the conjunction of the bindings it contains, and if we let $\mathbf{f} = \Psi_F(f)$ and $\mathbf{r} = \Psi_R(r)$, we can write this in the more transparent form

$$\Psi\Big( \bigwedge_i f_i/r_i \Big) = \sum_i \mathbf{f}_i \otimes \mathbf{r}_i .$$

The interpretation of the activity of binding units in the tensor product representation depends on the interpretation of the filler and role units. If the filler or role representations are local, then each filler or role unit individually represents a particular filler or role, respectively. If the filler or role representation is distributed, the activation of an individual node may indicate the presence of an identifiable feature in the entity being represented. This was true of the example given in Section 1.1: each of the output units represents a phonetic feature in the phonetic segment output by the network. For expository simplicity, we can consider a local representation to be one where a given "feature" is present in exactly one represented object, and a given object possesses exactly one "feature." Then if the binding unit $\tilde{b}_{\phi\rho}$ is active in the tensor product representation of a structure $s$, the interpretation is that the feature represented by $\tilde{f}_\phi$ is present in a filler of a role possessing the feature $\tilde{r}_\rho$. In this sense, $\tilde{b}_{\phi\rho}$ *represents the conjunction of the features represented by* $\tilde{f}_\phi$ *and* $\tilde{r}_\rho$.[2] By using the tensor product representation recursively, we can produce conjunctions of more than two features; this will be considered in Section 3.7.3.

## 2.3. Previous representations and special cases of tensor product representation

Section 3 analyzes the general properties of the tensor product representation. Before proceeding to this general analysis, it is useful to examine a number of special cases of the tensor product representation because these turn out to include nearly all previous cases of connectionist representations of structured objects.

### 2.3.1. *Role decompositions*

The examples of previous connectionist representations of structured objects that we shall consider employ only a few role decompositions.

[2] For a more precise formulation, consider a simple case where the activity of unit $\tilde{f}_\phi$ is 1 or 0, and indicates the truth value of the proposition "there exists $x$ among the represented objects such that the predicate $\tilde{f}_\phi$ holds of $x$"; and suppose $\tilde{r}_\rho$ can be similarly interpreted. Then $\tilde{b}_{\phi\rho}$ indicates the truth value of the proposition "there exists $x$ among the represented objects such that both predicates $\tilde{f}_\phi$ and $\tilde{r}_\rho$ hold of $x$." If this is true of $n$ different values of $x$, in the superposition representing the structure as a whole, the value of $\tilde{b}_{\phi\rho}$ will be $n$.

**Definition 2.11.** Suppose $S$ is the set of strings of length no more than $n$ from an alphabet $A$. Let $F = A$, and let $R = \{r_i\}_{i=1}^{n}$, where $r_i$ is the role "occupies the $i$th position in the string." Then $F/R$ is the *positional role decomposition* of $S$.

This is the example given above in Section 2.2, in which the string *aba* is represented by bindings $\{a/r_1, b/r_2, a/r_3\}$. This decomposition is finite, has single-valued roles, and is faithful. This decomposition is the most obvious one, and the one most often used in previous connectionist systems.

The positional decomposition has an obvious extension to the case of finite strings of arbitrary length, where the set of roles becomes infinite; I will treat this as the case of the above definition with $n = \infty$. In the infinite case the decomposition is still faithful, still has single-valued roles, and is still finite, since the strings are all of finite length. The infinite case will be used later to explore saturation of the tensor product representation.

There is a less obvious role decomposition of strings that is used, as we shall shortly see, to advantage by Rumelhart and McClelland [33]; it forms the basis of their "Wickelfeature" representation. (The properties of this role decomposition are crucial to many of the criticisms of this model presented in [17, 27, 28].)

**Definition 2.12.** Suppose $S$ is the set of strings of length no more than $n$ from an alphabet $A$. Let $F = A \cup \{<\ ,\ >\}$, where $<$ and $>$ are two new symbols meaning "left string boundary" and "right string boundary" respectively. Let $R = \{r_{x\_y} \mid x \in F,\ y \in F\}$, where $r_{x\_y}$ is the role "is immediately preceded by $x$ and immediately followed by $y$." $F/R$ is a role decomposition of $S$ called the 1-*neighbor context decomposition*.

Under this decomposition, the string *aba* becomes the set of bindings $\{a/r_{<\_b}, b/r_{a\_a}, a/r_{b\_>}\}$. This decomposition does not have single-valued roles and is not faithful if $n \geq 4$ (the strings $a^3$ and $a^4$ can't be distinguished). There is an obvious generalization to the $k$-neighbor context decomposition: this is faithful if $n < 2k + 2$.[3]

There are also obvious generalizations of the 1-neighbor context decomposition to differing size contexts on the left and right. A special case is the representation of pairs, say strings with $n = 2$, where the roles are $R = \{r_{\_x} \mid x \in F\}$: the right-neighbor context. The pair *ab* is represented as the single binding $a/r_{\_b}$. This role decomposition, we shall see, is used in a powerful technique called *conjunctive coding*.

[3] This decomposition gives the initial and final substrings of length up to $2k$, and all internal substrings of length $2k + 1$. These substrings uniquely determine strings of length no more than $2k + 1$. The strings $a^{2k+1}$ and $a^{2k+2}$ can't be distinguished, however, so the decomposition is not faithful if $n > 2k + 1$.

While it is true that the positional role decomposition is more faithful than context decompositions for the representation of a *single* structure, it turns out that if multiple structures are to be simultaneously represented, the positional decomposition can be *less* faithful than the context decomposition. Suppose we are to represent the conjunction of *ab* and *cd* by superimposing the representation of the two pairs. What gets represented is the union of the binding sets of the two structures. In the case of positional roles, this union is $\{a/r_1, b/r_2, c/r_1, d/r_2\}$; now it is impossible to distinguish what is being represented from the conjunction of *ad* and *cb*. However, with the right-neighbor context decomposition, the union of the binding sets is $\{a/r_{\_b}, c/r_{\_d}\}$, which is not at all confusable with the conjunction of *ad* and *cb*. With context decompositions confusions can of course also result; these decompositions are not even faithful for representing single structures, when the same fillers appear multiple times in the same context.

An additional virtue of context decompositions is that they give rise to connectionist representations that give the network direct access to the kind of information needed to capture the regularities in many context-sensitive tasks; we shall discuss this below for the specific example of the Rumelhart and McClelland [33] model.

### 2.3.2. *Connectionist representations*

Having discussed a few of the role decompositions that have been used in connectionist representations of structures, we can now consider a number of examples of such representations. These are grouped according to the degree of locality in the representations of roles and fillers; we therefore start by distinguishing local and distributed connectionist representations in general, and then examine the degree of locality of various existing representations of structured objects.

#### 2.3.2.1. *Local and distributed representations*

Local representations dedicate an individual processor to each item represented. In terms of the vector space of network states, these individual processors correspond to the members of the distinguished basis. Thus:

**Definition 2.13.** Let $\Psi$ be a connectionist representation of a set $X$ in a vector space $V$ with distinguished basis $\{\hat{\mathbf{v}}_i\}$. $\Psi$ is a *local representation* iff it is a one-to-one mapping of the elements of $X$ onto the set of basis vectors $\{\hat{\mathbf{v}}_i\}$.

A connectionist representation that is not a local representation is a *distributed representation*.

(For a more refined analysis of this distinction, see [43].)

#### 2.3.2.2. *Purely local representations of symbolic structures*

The first special case of the tensor product representation is the most local one.

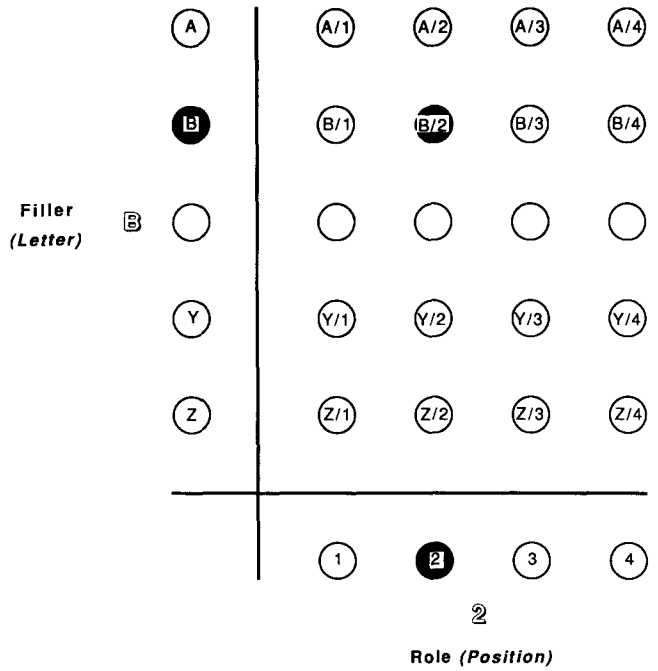


Fig. 5. A purely local tensor product representation of four-letter strings.

**Definition 2.14.** Let $\Psi_{F/R}$ be the tensor product representation of $S$ induced by a role decomposition $F/R$ of $S$ and two connectionist representations $\Psi_F$ and $\Psi_R$. Then $\Psi_{F/R}$ is a *purely local tensor product representation* if $\Psi_F$ and $\Psi_R$ are both local representations.

This case is illustrated for the representation of strings in Fig. 5. If the filler and role patterns both involve the activity of only a single processor, then the tensor product pattern representing their binding will also involve only a single unit. In other words, if $\Psi_F$ and $\Psi_R$ are both local representations, then $\Psi_{F/R}$ is a local representation of individual bindings.

Purely local tensor product representations have been used along with the positional role decomposition of strings in many connectionist models; for example:

- As was already mentioned in Section 1.1 and illustrated in Fig. 1, NETtalk [35] uses the purely local representation of Fig. 5 to represent seven-letter input strings.
- The interactive activation model of the perception of letters in words [23, 32] uses the representation shown in Fig. 5 for representing four-letter strings, at its intermediate or "letter" level of representation. This too is a purely local tensor product representation.

- The TRACE model of speech perception [21] uses a purely local representation of strings of phonemes, although some of the positional roles involve overlapping time intervals.
- Fanty's [8] parser uses a purely local tensor product representation involving a positional role decomposition of trees.
- Feldman's [9] connectionist system for visual processing uses a representation that includes the tensor product of a local representation for visual features (including color, size, and shape) and a local representation for position in the visual field.

In many of these models, the local representation of roles is not made explicit in the usual description of the model, but is rather implicit in the structure of the representation. The same is true of the semi-local case we take up next.

2.3.2.3. *Semi-local representations of symbolic structures*
The next most local special case is this.

**Definition 2.15.** Let $\Psi_{F/R}$ be the tensor product representation of $S$ induced by a role decomposition $F/R$ of $S$ and two connectionist representations $\Psi_F$ and $\Psi_R$. If $\Psi_F$ is a distributed representation and $\Psi_R$ is a local representation, then $\Psi_{F/R}$ is a *semi-local tensor product representation* or a *role register representation*.

If the filler representation is a distributed pattern and the role representation involves the activity of a single unit, the result is a copy of the filler pattern in a pool of units dedicated to the role: see Fig. 6.

Semi-local tensor product representations have been widely used in conjunction with positional role decompositions:

- The letter perception model [23, 32] uses a semi-local representation of letters at its lowest or "letter feature" level; this is the example shown in Fig. 6. A set of units is dedicated to the representation of the first letter's features; a letter is represented as a pattern of activity over these units, where each unit indicates whether a particular line segment is or is not present in the first letter. There are identical copies of this "first letter register" for the second, third, and fourth letter.
- An early version of NETtalk (Charles Rosenberg, unpublished communication, 1985) used a semi-local representation for the input string: the $i$th letter was represented by a pattern of activity over a set of units dedicated to the $i$th position, and each unit indicated whether a particular orthographic feature (e.g., closed loop, ascending line) was present in that letter.
- In Hinton's [13] semantic net model, relationships of the form $R(x, y)$ (e.g., *has_color*(*clyde*, *grey*)), are represented by placing three distributed

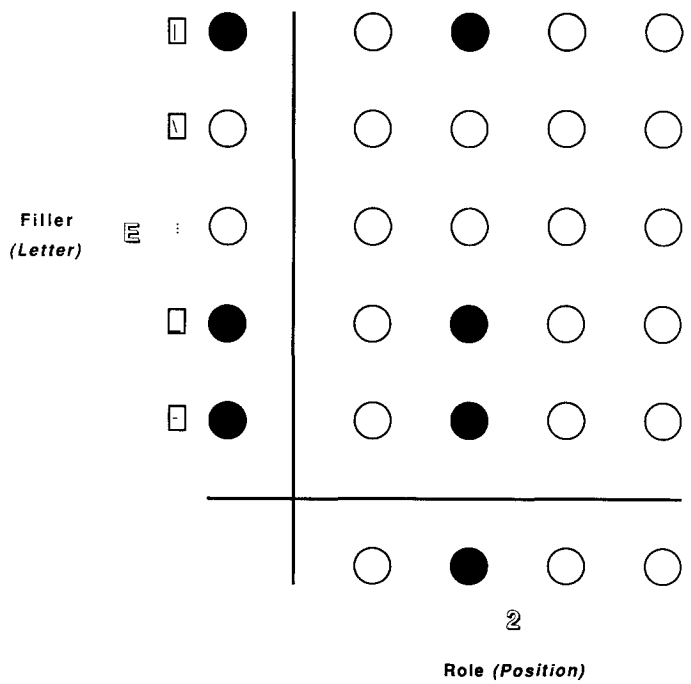


Fig. 6. A semi-local tensor product representation of four-letter strings.

patterns of activity representing the fillers of the roles $R$, $x$, and $y$ in pools dedicated to those roles. (There is an additional pool as well.)

- The model of Riley and Smolensky [29] that answers qualitative questions about a fixed simple electric circuit also uses a semi-local representation. Each role is a circuit variable (e.g., the current, or the resistance of one of the resistors) and the fillers are the qualitative values *increases*, *decreases*, *stays_constant*. Each filler is represented as a small pattern in a pool of two units dedicated to the corresponding role.
- Touretzky and Hinton's [46] connectionist production system interpreter uses productions with two symbolic triples on the condition side; each triple is represented by a pattern of activity in a separate pool of units. (The representation of the triples themselves are considered in the next section.)
- The McClelland and Kawamoto [22] model that learns to assign case to the nouns appearing in a standard sentence frame uses a semi-local representation of its input. Each input is an instance of the frame: *The* $N_1$ *V the* $N_2$ *with the* $N_3$. The roles here are the three nouns and the verb, and each filler is represented by a pattern of activity in a pool of units dedicated to the corresponding role.

2.3.2.4. *Fully distributed representations of symbolic structures*
Now we come to the most distributed case:

**Definition 2.16.** Let $\Psi_{F/R}$ be the tensor product representation of $S$ induced by a role decomposition $F/R$ of $S$ and two connectionist representations $\Psi_F$ and $\Psi_R$. If $\Psi_F$ and $\Psi_R$ are both distributed representations, then $\Psi_{F/R}$ is a *fully distributed tensor product representation.*

Examples of fully distributed representations include the following:

- A visually transparent example of a fully distributed tensor product representation using the positional role decomposition was given in Fig. 3. The patterns representing roles here are examples of *coarse coding* representations described in Hinton, McClelland, and Rumelhart [15]. It is traditional to focus on the numerous positions (roles) that activate a particular role unit (its "receptive field"); the formulation here focuses on the numerous role units activated by a particular positional role. These are merely two perspectives on the many-to-many mapping between positions and units.
- The McClelland and Kawamoto [22] model mentioned earlier can be viewed as using a fully distributed representation of the output. Each output is a set of bindings of noun fillers to the case-frame slots of the verb. This output can be viewed as having roles like *loves-agent*, *loves-patient*, *eat-instrument*, *break-patient*, and so on; these roles can in turn be viewed as structured objects with two sub-roles: *verb* and *case-role*. The patterns representing the overall roles are the tensor product of a distributed pattern representing the verb (built from semantic verb features) and a local representation of the case-role. The representation of the overall roles is thus semi-local. The representation of the output as a whole is the tensor product of this distributed (albeit semi-local) representation of the roles and a distributed representation of the fillers (built of semantic features of nouns). This is an example of the kind of recursive tensor product representation that will be discussed in Section 3.7.3. Because of this recursive structure, the output units in this model represent three-way conjunctions of features for nouns, verbs, and semantic roles. (The "features" of semantic roles used in the model are of the local type mentioned in Section 2.3.2.4: they are in one-to-one correspondence with the semantic roles. A more distributed version of this model would employ real features of semantic roles, where each semantic role is a distributed pattern of features. Then the roles in the output as a whole would have fully distributed representations instead of semi-local ones.)
- An example of a fully distributed representation employing the 1-neighbor context decomposition is the Rumelhart and McClelland [33] model that

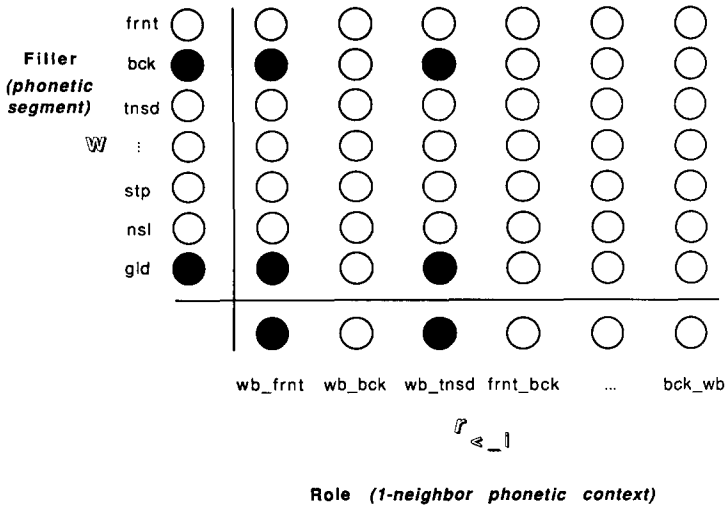


Fig. 7. The principal representation used in Rumelhart and McClelland [33] for phonetic strings. The abbreviations used are *wb = word_boundary*, *frnt = front*, *bck = back*, *tnsd = tensed*, *stp = stop*; *nsl = nasal*; *gld = glide*.

learns to form the past tense of English verbs; see Fig. 7. In this model, elements of $S$ are strings of phonetic segments. The word "weed" corresponds to the string $[w][i][d]$ which has the bindings $\{w/r_{<\_i}, i/r_{w\_d}, d/r_{i\_>}\}$. The representation of this string is thus

$$\mathbf{w} \otimes \mathbf{r}_{<\_j} + \mathbf{i} \otimes \mathbf{r}_{w\_d} + \mathbf{d} \otimes \mathbf{r}_{i\_>} .$$

The filler vectors (e.g. $\mathbf{w}$) are distributed patterns over a set of units representing phonetic features (e.g., *rounded*, *front*, *stop*). The role vectors (e.g. $\mathbf{r}_{<\_i}$) are patterns of activity over a set of units each of which represents the conjunction of a feature of the left neighbor ($<$) and a feature of the right neighbor ($i$). (In this model, both $<$ and $>$ possess the single feature *word_boundary*.) As in the previous example, since the roles are composite objects, they are in fact themselves further decomposed into sub-roles. The pair of phonetic segments defining the context is decomposed using the right-neighbor context decomposition, and the pattern representing the role $r_{a\_b}$ is the tensor product of patterns of phonetic features for $[a]$ and $[b]$. To reduce the number of units in the network, many of the units arising in this further decomposition of the roles were in fact discarded. The overall structure of the representation of the roles can still be productively viewed as a tensor product from which some units have been thrown away.

- Touretzky and Hinton's [46] representation of triples of letters can be viewed as the same sort of third-order tensor product as in the last example, but in which even more binding units are discarded. Their representation involves a set of units $\alpha = 1, \ldots, N$, each of which responds to three groups of letters $(L_\alpha^{(1)}, L_\alpha^{(2)}, L_\alpha^{(3)})$: unit $\alpha$ is active in the representation of $(l^{(1)}, l^{(2)}, l^{(3)})$ iff $l^{(i)} \in L_\alpha^{(i)}$ for $i = 1, 2$ and 3. To relate this to the tensor product representation, imagine three pools of $N$ units, one pool for each letter in the triple. In the $i$th pool, unit $\tilde{u}_\alpha^{(i)}$ is active iff $l^{(i)} \in L_\alpha^{(i)}$: each letter is represented by a pattern in the corresponding pool. Create a binding unit for each triple of units, one from each pool; it is active iff the corresponding three units are active. This is the tensor product representation of the triples induced by the 1-neighbor context decomposition, with the roles further decomposed by the right-neighbor context decomposition, as in the previous example. Now if we throw out all the binding units corresponding to $(\tilde{u}_\alpha^{(1)}, \tilde{u}_\beta^{(2)}, \tilde{u}_\gamma^{(3)})$, except the $N$ "diagonal" ones corresponding to $(\tilde{u}_\alpha^{(1)}, \tilde{u}_\alpha^{(2)}, \tilde{u}_\alpha^{(3)})$, we get Touretzky and Hinton's representation.
- Hinton [13] suggested an extension to his implemented model using fully distributed representations in which each unit represented a conjunction of features of an object, a relation, and a role.
- Derthick's μKLONE system [2] and Dolan and Dyer's story understanding system [4, 5] both use the fully distributed form of tensor product representation.
- Touretzky and Geva's DUCS system [45] uses fully distributed filler/role bindings that, while currently lacking a mathematical basis like that of tensor product representations, have many of their desirable properties, while requiring potentially far fewer units.

Having mentioned Rumelhart and McClelland's [33] use of context decompositions, it is worth elaborating on remarks of Section 2.3.1 about the advantages of context decompositions over simpler positional decompositions. Many regularities in language depend on the context in which a constituent finds itself, rather than its absolute position. This is particularly true in phonology; the regularities that must be learned in order to form the past tenses of English verbs typically depend on neighbor relations: for example, the rule for "regular" verbs involves replacing $x/r_{y\_>}$ by the bindings $\{x/r_{y\_\hat{}}, \hat{}/r_{x\_d}, d/r_{\hat{}\_>}\}$ if $x$ has feature *dental*, (*weed* → *weeded*); otherwise, by the bindings $\{x/r_{y\_d}, d/r_{x\_>}\}$ if $x$ has feature *voiced*, (*buzz* → *buzzed*) or by the bindings $\{x/r_{y\_t}, t/r_{x\_>}\}$ if $x$ does not have feature *voiced*(*bus* → *bussed*). Thus the featural representation of phonetic segments together with the context decomposition of the string provides the network with just the kind of representation of phonetic strings that it needs in order to learn the regularities characterizing this task (a point elaborated in detail in [17]).

### 2.4. Relations among purely local, semi-local, and fully distributed representations

Purely local, semi-local and fully distributed representations look quite different on the surface. Are they really as different as they seem? According to the definitions, the only difference is the relation between the representation vectors and the distinguished basis vectors indicating the individual processing units. Does this really matter?

As discussed at length in Smolensky [37], the answer depends on the dynamics driving the connectionist network, and not solely on the representations themselves. If the dynamics is linear, so that the activity of every unit is exactly a weighted sum of the activity of its neighbors in the network, then networks using purely local, semi-local and fully distributed representations will have exactly isomorphic behavior, subject to a few qualifications. Under the linear transformations that map these three cases into each other, locality is not preserved, so that local damage to the networks will have different effects, and what can be learned via the usual local connectionist learning procedures will be different. If the network contains nonlinear units, the isomorphism fails. Also, assuming finite networks, the local case accommodates only a fixed, finite set of fillers and roles; the semi-local case allows an unlimited number of fillers but only a finite set of roles. The fully distributed case, however, can accommodate an infinite set of fillers and roles in a finite network, since the vectors representing both the roles and fillers can be arbitrary activity patterns drawn from vector spaces which, while having finite dimension, contain a continuous infinity of distinct vectors; this will be discussed further in Section 3.2.

## 3. Tensor Product Representation: Properties

In Section 2, I defined the tensor product representation and showed how a number of representations used in previous connectionist models are various special cases of the tensor product representation. In this section, I will discuss a number of general properties of this representation. The case of interest is fully distributed representation; while some of the results apply also to the more localized special cases, in these cases they become rather trivial.

### 3.1. Unbinding

Until now I have ignored a crucial and obvious question: if the representations of all the variable bindings necessary for a particular structure are superimposed on top of each other in a single set of binding units, how can we be sure the binding information is all kept straight? In this section we explore this question via the *unbinding* process: taking the tensor product representation for a complex structure and extracting from it the filler for a particular role. Under what conditions can we perform this unbinding operation accurately?

**Theorem 3.1.** *Let $\Psi_{F/R}$ be a tensor product representation induced by a role decomposition with single-valued roles. Suppose the vectors representing the roles bound in a structure s are linearly independent. Then each role can be unbound with complete accuracy: for each bound role $r_i$ there is an operation which takes the vector $\Psi_{F/R}(s)$ representing s into the vector $\mathbf{f}_i$ representing the filler $f_i$ bound to $r_i$.*

**Proof.** If the role vectors $\{\mathbf{r}_i\}$ being used are linearly independent, then they form a basis for the subspace of $V_R$ that they span. To this basis there corresponds a *dual basis* $\{U_i\}$ [19, p. 82]. Each element in this dual basis is a linear mapping from $V_R$ into the real numbers with the property that

$$U_i(\mathbf{r}_j) = \delta_{ij} = \begin{cases} 1 & \text{if } i = j\,, \\ 0 & \text{if } i \neq j\,. \end{cases}$$

That is, $U_i$ maps the single role vector $\mathbf{r}_i$ to 1 and all other role vectors to 0. If we make use of the canonical inner product on the vector space $V_R$, then the dual vector $U_i$ can be expressed as the operation of taking the inner product with respect to some vector $\mathbf{u}_i$ in $V_R$:

$$U_i(\mathbf{v}) = \mathbf{v} \cdot \mathbf{u}_i$$

for all $\mathbf{v}$ in $V_R$. Call $\{\mathbf{u}_i\}$ the *unbinding vectors* for roles $\{r_i\}$. Now let $\mathbf{s}$ be the tensor product representation of a structure in which the roles $\{r_i\}$ are bound to the fillers $\{f_i\}$. Then we can extract $f_i$ from $\mathbf{s}$, or unbind $r_i$, by taking a partial inner product of $\mathbf{s}$ with the unbinding vector $\mathbf{u}_i$:

$$\begin{aligned} \mathbf{s} \cdot \mathbf{u}_i &= \left(\sum_j \mathbf{f}_j \otimes \mathbf{r}_j\right) \cdot \mathbf{u}_i \\ &:= \sum_j \mathbf{f}_j (\mathbf{r}_j \cdot \mathbf{u}_i) = \sum_j \mathbf{f}_j \delta_{ij} = \mathbf{f}_i\,. \end{aligned}$$ □

Connectionist algorithms for computing the unbinding vectors will be discussed in Section 3.4.2.

**Definition 3.2.** The procedure defined in the preceding proof is the *exact unbinding procedure.*

Let unbinding of role $r_i$ be performed as in the previous proof, but in place of the unbinding vector $\mathbf{u}_i$ use the role vector $\mathbf{r}_i$ itself. This is the *self-addressing unbinding procedure.*

Unlike the exact unbinding procedure, the self-addressing unbinding procedure is defined for any set of role vectors, even if they are not linearly independent.

**Theorem 3.3.** *Suppose the self-addressing procedure is used to unbind roles. If the role vectors are all orthogonal, the correct filler pattern will be generated, apart from an overall magnitude factor. Otherwise, the pattern generated will be a weighted superposition of the pattern of the correct filler,* $\mathbf{f}_i$*, and all the other fillers,* $\{\mathbf{f}_j\}_{j \neq i}$*. In this superposition, the weight of each erroneous pattern* $\mathbf{f}_j$ *relative to the correct pattern* $\mathbf{f}_i$*, the* intrusion of role $j$ into role $i$*, is*

$$\frac{\mathbf{r}_i \cdot \mathbf{r}_j}{\|\mathbf{r}_i\|^2} = \cos \theta_{ji} \frac{\|\mathbf{r}_j\|}{\|\mathbf{r}_i\|}$$

*where* $\theta_{ji}$ *is the angle between the vectors* $\mathbf{r}_j$ *and* $\mathbf{r}_i$*.*

**Proof.**

$$\mathbf{s} \cdot \mathbf{r}_i = \left(\sum_j \mathbf{f}_j \otimes \mathbf{r}_j\right) \cdot \mathbf{r}_i = \sum_j \mathbf{f}_j (\mathbf{r}_j \cdot \mathbf{r}_i)$$

$$= (\mathbf{r}_i \cdot \mathbf{r}_i)\mathbf{f}_i + \sum_{j \neq i} (\mathbf{r}_j \cdot \mathbf{r}_i)\mathbf{f}_j \, .$$

In this weighted superposition, the ratio of the coefficient of each incorrect filler $\mathbf{f}_j$ to that of the correct filler $\mathbf{f}_i$ is

$$\frac{\mathbf{r}_j \cdot \mathbf{r}_i}{\mathbf{r}_i \cdot \mathbf{r}_i} \, .$$

The denominator is $\|\mathbf{r}_i\|^2$ and the numerator is $\cos \theta_{ji} \|\mathbf{r}_j\| \, \|\mathbf{r}_i\|$, giving the claimed result. □

Note that if two roles have very similar representations, there can be substantial confusion about what their respective fillers are. The next section provides a quantitative result on the intrusion of one role on another. If the role vectors are linearly independent, the exact unbinding procedure can be used to eliminate intrusions, but the unbinding vectors must be computed.

Since the tensor product binding representation is symmetric between role and filler, the unbinding procedures given above can also be used to retrieve a role pattern from the filler pattern to which it is bound. While there is no asymmetry between role and filler in the representation of a single binding, an asymmetry may however result from the combination of many bindings in the representation of a structured object. For while role decompositions often involve single-valued roles, it is uncommon to encounter single-valued fillers. Thus while there will often be a unique filler indexed by a given role, there will sometimes be several roles associated with a single filler. In the latter case, an

unbinding that is performed using the filler pattern as an index will generate the superposition of all the role vectors bound to that filler.[4]

### 3.2. Graceful saturation

Like a digital memory with $n$ registers, a connectionist system that uses $n$ pools of units to represent a structure with $n$ roles has a discrete saturation point. Structures with no more than $n$ roles filled can be represented precisely, but for larger structures some information must be omitted entirely. The form of saturation characteristic of connectionist systems (e.g., connectionist memories) is less discrete than this; this is one aspect of the "graceful degradation" advertised for connectionist systems.

Aspects of the graceful degradation notion can be formally characterized as follows:

**Definition 3.4.** Let $F/R$ be a role decomposition of $S$. A connectionist representation $\Psi$ of $S$ has *unbounded sensitivity* with respect to $F/R$ if for arbitrarily large $n$,

$$\Psi\left(\bigwedge_{i=1}^{n} f_i/r_i\right)$$

varies as $f_i$ varies, for each $i = 1, 2, \ldots, n$.

If for sufficiently large $n$ the representation of structures containing $n$ filler/role bindings is not faithful, then $\Psi$ *saturates*.

If $\Psi$ saturates and has unbounded sensitivity then $\Psi$ possesses *graceful saturation*.

The tensor product representation, unlike local and role register representations, can exhibit graceful saturation. To show this, I now consider an example that also illustrates how fully distributed tensor product representations can be used to represent an infinite number of roles in a finite-dimensional vector space corresponding to a finite connectionist network.

**Theorem 3.5.** *Suppose $S$ is the set of finite strings (with no upper bound on length), and let $\{r_i\}_{i=1}^{\infty}$ be the positional roles. Let the vectors $\{\mathbf{r}_i\}_{i=1}^{\infty}$ be unit*

[4] Related to unbinding are Mozer's *pullout networks* [25]. These networks take an input that represents a mixture of several coherent objects and "pulls out" the vector representing a single object, suppressing the representation of the others. This is done by setting up connections encoding a set of constraints that define what it means for a vector to represent a coherent object, and using relaxation to settle on a single coherent representation. At least in this normal usage, pullout networks solve a problem that is related to but different from unbinding. The starting point of unbinding is a representation of a single coherent object (structure); the problem is, given a role in the structure, find what fills it (or vice versa).

*vectors in N-dimensional space, randomly chosen according to the uniform distribution. Then this tensor product representation possesses graceful saturation. The expected value of the magnitude of the intrusion of role i into role j is proportional to* $N^{1/2}$. *The number of bindings n that can be stored before the expected total magnitude of intrusions equals the magnitude of the correct pattern increases as* $N^{1/2}$.

**Proof.** Since all role vectors have unit length, from Theorem 3.3, the expected value of the magnitude of the intrusion is

$$\mathrm{EI} = \frac{1}{V_{N-1}} \int_0^{\pi} |\cos \theta_{ji}| \, V(\theta_{ji}) \, \mathrm{d}\theta_{ji} \, .$$

Here $V_{N-1}$ is the $N-1$-dimensional volume of the unit sphere in $N$-space, and $V(\theta_{ji})$ is the volume of the subset of the unit sphere in $N$-space consisting of all vectors having angle $\theta_{ij}$ with the vector $\mathbf{r}_i$. This subset is in fact a sphere in $N-1$-space with radius $\sin \theta_{ji}$. To see this, choose a Cartesian coordinate system in $N$-space in which the first coordinate direction lies along $\mathbf{r}_i$. Then the first coordinate $x_1$ of all points in the subset is $\cos \theta_{ji}$. Since all points lie on the unit sphere, we have

$$1 = \sum_{i=1}^{N} x_i^2 = \cos^2 \theta_{ji} + \sum_{i=2}^{N} x_i^2$$

which implies

$$\sum_{i=2}^{N} x_i^2 = 1 - \cos^2 \theta_{ji} = \sin^2 \theta_{ji} \, .$$

Thus the subset is a sphere in $N-1$-space with radius $\sin \theta_{ji}$. Therefore

$$V(\theta_{ji}) = V_{N-2} \sin^{N-2} \theta_{ji} \, .$$

Thus the expected intrusion is

$$\mathrm{EI} = \frac{V_{N-2}}{V_{N-1}} \, 2 \int_0^{\pi/2} \sin^{N-2} \theta \cos \theta \, \mathrm{d}\theta$$

$$= \frac{V_{N-2}}{V_{N-1}} \, 2 \int_0^{1} z^{N-2} \, \mathrm{d}z = \frac{V_{N-2}}{V_{N-1}} \, \frac{2}{N-1}$$

(using the substitution $z = \cos \theta$ which implies $\mathrm{d}z = -\sin \theta \, \mathrm{d}\theta$). The ratio of volumes of spheres of successive dimensions $V_{N-2}/V_{N-1}$ is a complex expres-

sion taking different forms depending on whether $N$ is odd or even (see the Appendix of [39]). Since these details are quite irrelevant to the general behavior as $N$ increases, we can look at the mean of two successive such ratios (using the geometric mean since the quantities are ratios) which is given by the simple expression[5]

$$\sqrt{(N-1)/2\pi}\,.$$

The result then is

$$\mathrm{EI} = \sqrt{\frac{2}{\pi(N-1)}}\,.$$

As claimed, the expected interference falls as $N^{-1/2}$.

For a structure involving $n$ bindings, the expected total magnitude of intrusions of all $\{r_j\}_{j\neq i}$ into $r_i$ is $(n-1)\mathrm{EI}$. This equals unity at

$$n = \sqrt{\frac{1}{2}\,\pi}\,(N-1)^{1/2} + 1$$

which increases as the square-root of $N$. □

The estimate of interference given in the preceding theorem is a very conservative one, since it computes the expected sum of the *absolute values* of all intrusions. In fact, for any given component of the desired filler, the errors caused by intrusions will be of both signs, producing a net error much smaller than the worst case analyzed above.

### 3.3. Continuous structures and infinite-dimensional representations

Certain structures are characterized by a continuum of roles. Strings, for example, have a natural extension to a continuum of positions. Examples of such continuous one-dimensional "strings" include speech input and motor output; a two-dimensional example is an image.

The tensor product representation extends naturally to the case of a continuum of roles. The representation of the conjunction of bindings extends naturally from the sum over a discrete set of bindings to an integral over a continuum of bindings.

[5] There is a rough calculation that suggests that, as the dimension $N$ grows, the expected inner product of unit-length role vectors should decrease with the square root of $N$. Suppose for the first $N$ role vectors we chose an orthonormal basis. For the next vector, suppose we choose one that is equidistant from all the others; an example is the vector whose components in the orthonormal basis are $C^{-1}(1, 1, \ldots, 1)$. In order for this vector to have unit length, the normalization constant $C$ must be $\sqrt{N}$. Now the inner product of this vector with any of the others is $C^{-1} = N^{-1/2}$.

**Definition 3.6.** Let $F/R$ be a role decomposition of $S$, not necessarily finite, and let $\mathrm{d}\mu(r)$ be a measure on $R$. Let $\mathrm{supp}_R(s)$ be the subset of $R$ containing roles which are bound in $s$, and suppose $F/R$ has single-valued roles. Suppose given the connectionist representations

$$\Psi_F : F \to V_F ; \qquad f \mapsto \mathbf{f} ,$$

$$\Psi_R : R \to V_R ; \qquad r \mapsto \mathbf{r}$$

and assume these functions are measurable with respect to $\mathrm{d}\mu(r)$. Then the corresponding tensor product representation of $S$ is

$$\Psi_{F/R}(s) = \int_{\mathrm{supp}_R(s)} \mathbf{f}(r) \otimes \mathbf{r} \, \mathrm{d}\mu(r) .$$

$\Psi_{F/R}(s)$ is defined only for those $s$ for which the integral is well-defined: $\mathrm{supp}_R(s)$ must be a measurable set and the integral must converge.

If the role decomposition is finite, and $\mathrm{d}\mu$ is counting measure, then this reduces to the previous definition of the tensor product representation.

In the case of a continuous string, we can take the roles to be $r(t)$ for a continuous time index $t$. For the measure we can use ordinary Lebesque measure on $t$. Then if each role is represented by a pattern $\mathbf{r}(t)$ and the fillers by the patterns $\mathbf{f}(t)$, the entire continuous string is represented by $\int \mathbf{f}(t) \otimes \mathbf{r}(t) \, \mathrm{d}t$. This representation of the continuous structure goes over exactly to the discrete case if it happens that the fillers are discrete step-functions of time. Suppose the filler $\mathbf{f}(t)$ is constant over the interval $[t_i, t_{i+1}]$ with value $\mathbf{f}_i$. Then the representation of the string is

$$\begin{aligned} \int_t \mathbf{f}(t) \otimes \mathbf{r}(t) \, \mathrm{d}t &= \sum_i \int_{t_i}^{t_{i+1}} \mathbf{f}(t) \otimes \mathbf{r}(t) \, \mathrm{d}t \\ &= \sum_i \int_{t_i}^{t_{i+1}} \mathbf{f}_i \otimes \mathbf{r}(t) \, \mathrm{d}t \\ &= \sum_i \mathbf{f}_i \otimes \int_{t_i}^{t_{i+1}} \mathbf{r}(t) \, \mathrm{d}t = \sum_i \mathbf{f}_i \otimes \mathbf{r}_i \end{aligned}$$

where the vector representing the discrete role for the time slot $[t_i, t_{i+1}]$ is the

integral of the vectors representing the time points in the slot:

$$\mathbf{r}_i := \int_{t_i}^{t_{i+1}} \mathbf{r}(t)\, \mathrm{d}t\,.$$

The representation of a continuous string can be visualized with the help of the example illustrated in Fig. 3, which shows a tensor product binding between a time and the energy level of a speech formant. The patterns representing the energy level and time are peaks centered at the values being represented; this can apply to continuous represented values as well as discrete ones. The pattern $\mathbf{r}_4$ representing time $i = 4$ (shown in Fig. 3) is a peak centered on the fourth role unit; a pattern $\mathbf{r}(4.2)$ representing time $t = 4.2$ would be derived by taking a peak on the continuous line centered at 4.2 and evaluating it at the integer values $i = 1, 2, \ldots, 5$. One can similarly generate patterns representing continuous energy levels $\mathbf{f}(t)$. As in the discrete case, the tensor product representation of the binding $f(t)/r(t)$ then becomes a two-dimensional peak centered at $(t, f(t))$, evaluated at points with integer coordinates. Superimposing the representation of the bindings for all $t$, we get the representation of the continuous string of energy levels: it resembles a smeared-out version of the graph of energy versus time, the activity of each unit in the grid of Fig. 3 being greater the closer it lies to the actual graph.

In the representation illustrated in Fig. 3, the role and filler vector spaces have finite dimensions (5 and 4, respectively). In such a case it is of course impossible for all the role vectors to be linearly independent; that would require an infinite-dimensional role vector space. The tensor product representation applies as well to infinite-dimensional vector spaces as to finite-dimensional ones. In that case the patterns representing roles (and possibly also fillers) would not be patterns defined by a finite number of values as shown in Fig. 3 but could rather be curves defined over a continuous segment. The peaked patterns representing energy levels and times could be smooth Gaussians over a fixed interval, with mean equal to the quantity being represented and with variance, say, some fixed value. Then the representation of each binding would be a two-dimensional smooth Gaussian with mean at the point with $x$- and $y$-coordinates equal to the time and energy values, respectively.

If the role space is infinite-dimensional, then so too will be the binding space. To view this space as the states of a connectionist network would require postulating an infinite number of units, one for each dimension of the space. The infinite-dimensional case is of interest not for computer simulation but for analysis; patterns which are functions of a continuous variable pose no particular difficulty for analysis relative to patterns which are finite-dimensional vectors.

It is significant that the tensor product representation extends so naturally to

continuous collections of roles, continuous sets of fillers, and vectors for representing roles and fillers that are continuous patterns. As I have argued elsewhere [41], an important characteristic of a number of connectionist networks is the existence of an underlying continuous model. Thus one indication that a connectionist representational scheme is well-motivated is that it has a natural continuous extension, even if particular simulation models take advantage only of the discrete case.

### 3.4. Connectionist mechanisms for binding and unbinding

The tensor product representation has so far been characterized mathematically, without any discussion of how such a representation might be set up and used in a connectionist system. In this section I consider first the creation of bindings and then I take up unbinding.

#### 3.4.1. *Parallel binding in connectionist systems*

The most immediate application of the tensor product representation is to models learning to map some structured input to structured output; for example, the surface form of a sentence to its parsed form. Here it is not the job of the network to set up the tensor product representations: in presenting the input/output pairs to the network during training, the modeler must convert the symbolic inputs and outputs to their vector representations, and this can be done directly by using the mathematical definition of the tensor product representation.

In more complex applications, a network might be so constructed as to internally perform variable binding via the tensor product. A convenient way to achieve this is to use so-called *sigma-pi* processing units [30]. Such a unit has a number of input sites at each of which connections from a number of other processors converge. For each site $\sigma$, the sigma-pi unit takes the *product* of all the inputs $\{I_{\sigma i}\}$ there; it then adopts as its value $\nu$ a weighted *sum* over all sites, with one weight $w_\sigma$ per site:

$$\nu = \sum_\sigma w_\sigma \prod_i I_{\sigma i}$$

Using sigma-pi units, tensor product binding can be easily achieved in a connectionist network: see Fig. 8. The network consists of a set of filler units $\tilde{f}_\phi$, a set of role units $\tilde{r}_\rho$, and set of binding units $\tilde{b}_{\phi\rho}$, one for each pair of filler and role units. Each binding unit is a sigma-pi unit with a single site with unit weight. Converging on the site of the binding unit $\tilde{b}_{\phi\rho}$ are two connections, one from $\tilde{f}_\phi$ and one from $\tilde{r}_\rho$. Then if the filler and role patterns **f** and **r** are set up on the filler and role units, the binding units will set up the representation of the binding **f**/**r**.

Fig. 9 shows a network equivalent to the one shown in Fig. 8. Here the

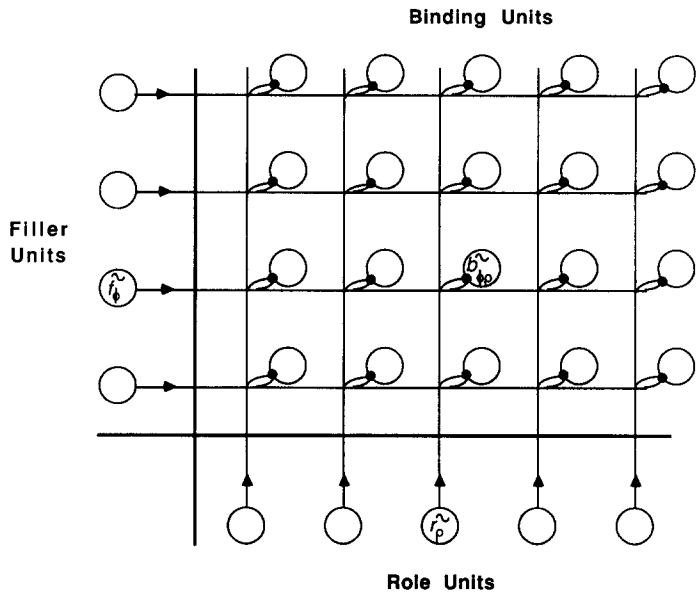


Fig. 8. A network using sigma-pi binding units to perform tensor product binding.

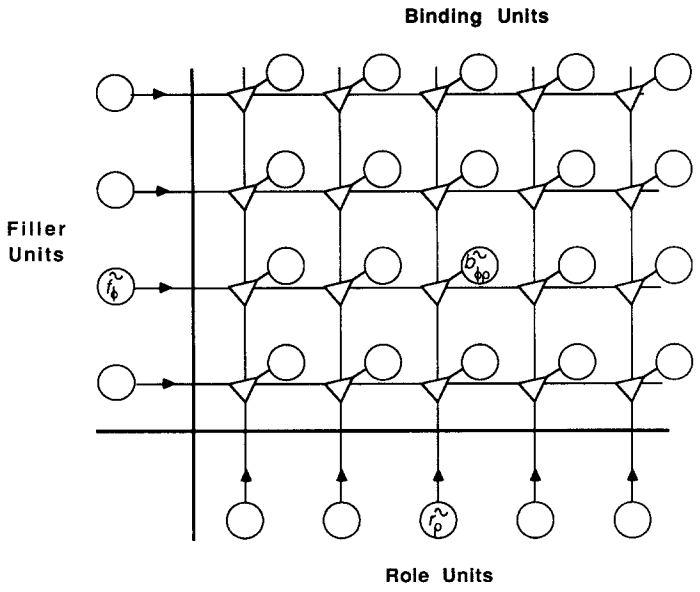


Fig. 9. A network using multiplicative junctions to perform tensor product binding.

product occurs not at the unit but at the junction of two connections; the two activities entering the triangular junction [12] from the filler and role units are multiplied together and the result is sent along the third line to the binding unit.

The representation of complex structures requires superimposing multiple filler/role bindings. There are two obvious ways of doing this: sequentially and in parallel. In the sequential case, one binding is performed at a time, and the binding units accumulate their activity over time. This can be achieved with the network shown in Fig. 8 if we use accumulating sigma-pi binding units obeying

$$\frac{dv}{dt} = \sum_{\sigma} w_{\sigma} \prod_{i} I_{\sigma i} .$$

Equivalently, serial binding can be performed by the network of Fig. 9 if the binding units accumulate activity over time.

In order to superimpose all $N$ bindings in parallel, we need to extend the network shown in Fig. 8, creating nodes $\{f_{\phi}^{(\sigma)}\}_{\sigma=1}^{N}$ and $\{r_{\rho}^{(\sigma)}\}_{\sigma=1}^{N}$: see Fig. 10, which illustrates the simplest case, $N = 2$. Now each sigma-pi binding unit has $N$ sites instead of one; each site has unit weight. Each site $\sigma$ on binding unit $\tilde{b}_{\phi\rho}$ receives a pair of connections from the nodes $\tilde{f}_{\phi}^{(\sigma)}$ and $\tilde{r}_{\rho}^{(\sigma)}$. Now we can bind $N$ pairs of roles and fillers in parallel. In the $\sigma$th filler pool we set up the pattern

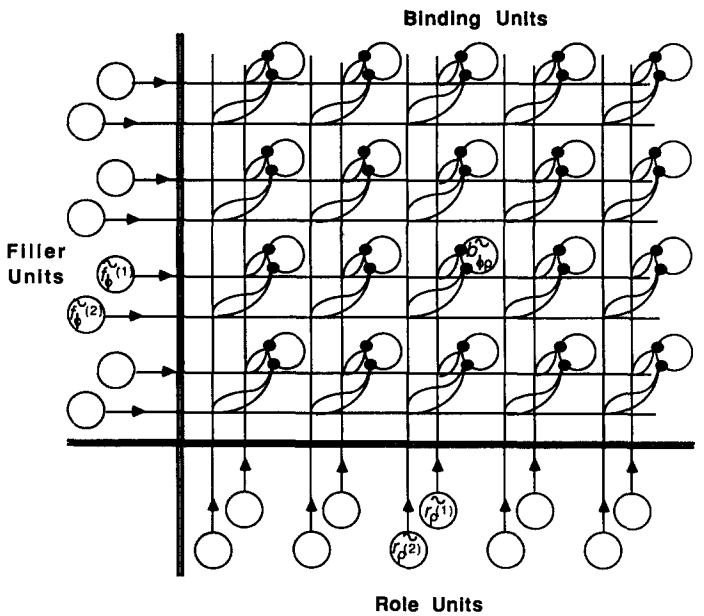


Fig. 10. An extension of the network of Fig. 8 that can perform two variable bindings in parallel.

$\mathbf{f}_\sigma$ representing $f_\sigma$ and on the $\sigma$th role pool we set up the pattern $\mathbf{r}_\sigma$ representing $r_\sigma$. The value of binding unit $\tilde{b}_{\phi\rho}$ is then

$$\tilde{b}_{\phi\rho} = \sum_\sigma 1 \prod \{\tilde{f}_\phi^{(\sigma)}, \tilde{r}_\rho^{(\sigma)}\} = \sum_\sigma (\mathbf{f}_\sigma)_\phi (\mathbf{r}_\sigma)_\rho \, .$$

The pattern of activity on the binding units is thus the correct tensor product representation of the structure. Fig. 11 is the equivalent of Fig. 10 using multiplicative junctions instead of sigma-pi units.

There is no need to perform *all* the binding serially or in parallel; the mechanisms of sequential and parallel combination of bindings are independent, and can be combined. If there are $N$ pools of filler and role units, $N$ bindings can be established in parallel, and if the binding units accumulate activity over time, further bindings can be added sequentially, up to $N$ at a time.

There are two senses in which bindings are occurring in parallel here. Bindings are *generated* in parallel, $N$ at a time; the *generation capacity* is sharply defined by $N$. At the same time, multiple bindings are being *maintained* in parallel; the binding units can simultaneously support multiple bindings superimposed on each other. The *maintenance capacity* of the representation is

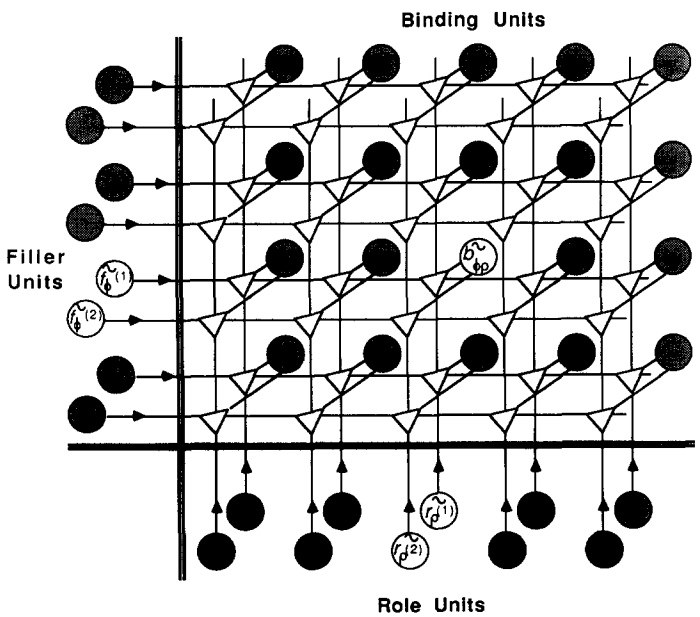


Fig. 11. An extension of the network of Fig. 9 that can perform two variable bindings in parallel. (Unlabelled units have been shaded to enhance readability.)

not sharply defined, due to the graceful saturation of the representation. The scale of the maintenance capacity is, however, set by $n$, the number of role units in each of the $N$ sets.

For the network shown in Fig. 10, the generation and maintenance capacities are independent; this contrasts with most existing connectionist systems. For example, the McClelland and Rumelhart letter perception model processes exclusively four letter words. Strings of length $n = 4$ can be represented; the maintenance capacity is precisely defined at four letters. The binding of all four letters to their positions are all performed in parallel; the generation capacity is also $N = 4$. If different roles correspond to different regions of a parallel network, as in local and semi-local representations, it is natural that these roles should all be sent activiation in parallel. If the different roles share a common set of units, as in fully distributed representations, there comes the space/time trade-off we have seen above: duplicate machinery to permit parallel binding, or wait while multiple bindings are performed serially.

It seems intuitive that the two binding capacities ought to be independent characteristics of the degree of parallelism in a processing system. In many human cognitive processes, for example, the generation capacity of binding appears to be much smaller than the maintenance capacity: $N \ll n$. In visual perception we are able to maintain rich percepts involving a huge number of bindings of properties to locations, but it turns out that at any one time (requiring approximately 50 msec) our visual systems can only establish the bindings for a small region of the visual field [47]. The large number of bindings that we maintain in parallel are generated a small fraction at a time through an extended sequential process. In discourse processing, syntactic and semantic processes seem to indicate that many constituents in complex structures are being maintained and processed in parallel, yet only a small fraction of these constituent/role bindings are generated at once. If one looks at the processing of small linguistic and/or visual items whose size fits within the binding generation capacity (e.g. four-letter words), the distinction between the generation and maintenance capacities does not assert itself. However, connectionist models of more complex, extended tasks such as reading whole passages must respect the distinction between these two aspects of parallelism; the tensor product representation offers a natural way to do so, because the machinery needed to create one binding can also create the others: there is no need to build separate hardware to create each binding, as is typically done with local or semi-local representations.

### 3.4.2. *Connectionist unbinding mechanisms*

The mathematics of the unbinding procedure was described in Section 3.1. It is easy to implement this procedure in a connectionist network; in fact, the network of Fig. 9 can be used for unbinding as well as for binding. We presume that the binding units are supporting a pattern of activity which is the tensor

product representation of a structure. To unbind role $r_i$, a pattern of activity is first set up on the role units: for the exact unbinding procedure, this pattern should be that of the unbinding vector $\mathbf{u}_i$; for the self-addressing unbinding procedure, the pattern should be $\mathbf{r}_i$. As a result of the activity in the role and binding units, a pattern of activity arises on the filler units. At each triangular junction, the activity of the connected role and binder units are multiplied together and sent to the connected filler unit, which adds up all the inputs it so receives (I assume, following [12], that the triangle junctions operate symmetrically, multiplying the activities arriving on any two lines and sending the product out along the third line.) Thus the activity of filler unit $\tilde{f}_\phi$ is

$$\tilde{f}_\phi = \sum_\rho \tilde{r}_\rho \tilde{b}_{\phi\rho} .$$

This is the correct activity to implement the unbinding procedures of Section 3.1. With the extended network shown in Fig. 11, $N$ roles can be unbound simultaneously.

This procedure has been defined for retrieving a filler from a role. By interchanging roles and fillers, it can also be used to retrieve a role from a filler, subject to the caveat of Section 3.1 about non-single-valuedness.

The unbinding vectors needed for the exact unbinding procedure can be stored in a network in a number of ways. Using an additional group of units, a local representation of roles for unbinding could be used, so that when it was desired to unbind a given role, the corresponding unit could be activated, and the weights on connections emanating from that unit could be used to set up the corresponding unbinding vector on the role units. Alternatively, the additional group of units could be used with a distributed representation; the actual vector representing $r_i$ could be set up on these units, and feedforward connections could then set up the corresponding unbinding vector $\mathbf{u}_i$ on the role units. It is always possible to perform this mapping linearly when the role vectors are linearly independent, which is also the condition required for the unbinding vectors to be defined in the first place. In fact, the Widrow–Hoff learning rule can be used to learn the weights necessary to map the $\{\mathbf{r}_i\}$ into the $\{\mathbf{u}_i\}$, provided a teacher can present the target vectors $\{\mathbf{u}_i\}$. Since the matrix of $\{\mathbf{u}_i\}$ is just the inverse of the matrix of $\{\mathbf{r}_i\}$, iterative matrix inversion algorithms implemented in a connectionist network can be used to compute the $\{\mathbf{u}_i\}$ from the $\{\mathbf{r}_i\}$. One simple way to do this is as follows (see also [18]). Using Widrow–Hoff learning, train a simple linear associator to map each $\mathbf{r}_i$ to the $i$th unit vector, i.e., to a local representation of roles with a unique unit active per role. When the $\mathbf{r}_i$ are linearly independent, this is always possible. Now, we make the connections in this linear associator bi-directional and symmetric. If a role vector $\mathbf{r}_i$ is placed on the input units of the associator, it will create a local pattern on the output units. If we send this activity

backwards along the same connections, then the new pattern set up on the input units will be exactly $\mathbf{u}_i$.[6]

### 3.5. Binding unit activities as connection weights

In Section 3.4.1 we discussed one way of generating the tensor product representation of a structure: sequentially representing individual filler/role pairs on the role and filler units, while each binding unit takes the product of the activities of its corresponding pair of role and filler units. These products then accumulate on the binding units as the individual pairs are presented. This procedure is formally identical to the *Hebbian learning procedure* for storing the associations between roles and corresponding fillers: each binding unit plays the role of the *connection* between a role and filler unit, and its activity plays the role of the weight or strength of that connection. Furthermore, the self-addressing unbinding mechanism described in Section 3.4.2 is formally identical to the use of the Hebbian weight matrix to associate a pattern over the role units with the corresponding pattern on the filler units.

This relationship between binding units and connections suggests avenues for further exploration, two of which will now be briefly described.

#### 3.5.1. *From Hebbian to Widrow–Hoff weights*

In Section 3.1 it was pointed out that the pattern needed for exact unbinding of role $r_i$, the unbinding vector $\mathbf{u}_i$, is not in general equal to the role vector $\mathbf{r}_i$; the retrieval and role patterns are equal only if the role vectors are orthonormal. This corresponds to a well-known property of the Hebbian weight matrix: associations will be correctly formed by the Hebbian learning procedure if and only if the input patterns are orthogonal. There is a more complex learning procedure than the Hebbian one which produces a matrix with better retrieval capability than the Hebbian matrix: the Widrow–Hoff [49] or delta rule [30]. This suggests replacing the Hebbian matrix corresponding to the tensor product representation with the Widrow–Hoff matrix. With this new representation, the self-addressing unbinding procedure would produce correct results as long as the role vectors are *linearly independent*: orthogonality is not required. Unfortunately, this Widrow–Hoff representation is considerably more difficult to write down, analyze, and actually construct in a connectionist network. For example, the Widrow–Hoff learning procedure, unlike the Hebbian one, requires repeated presentations of the set of items to be stored.

[6] The weights $\{W_{kj}\}$ needed to map the $\{\mathbf{r}_i\}$ to the output basis vectors satisfy $\Sigma_j W_{kj}(\mathbf{r}_i)_j = \delta_{ki}$. In other words, $\mathbf{W}$ is the inverse of the matrix of role vectors. The $k$th row of $\mathbf{W}$ is thus just the unbinding vector $\mathbf{u}_k$; i.e., if we define $(\boldsymbol{u}_k)_j = W_{kj}$, then $\mathbf{u}_k \cdot \mathbf{r}_i = \delta_{ki}$ which is the defining property of the unbinding vectors. When $\mathbf{r}_i$ is sent through $\mathbf{W}$, it sets up the $i$th unit vector, and when this is sent back through the same connections, the $j$th activity is $\Sigma_k W_{kj}(\delta_{ki}) = W_{ij} = (\mathbf{u}_i)_j$. That is, the vector $\mathbf{r}_i$ has been replaced by the vector $\mathbf{u}_i$.

A relaxation process could be used to do Widrow–Hoff binding [D. Rumelhart, personal communication], but it would require that all filler/role pairs be simultaneously presented to the relaxation network. This would break the independence, discussed in Section 3.4.1, of the generation and maintenance capacities for binding.

3.5.2. *Relation to Connection Information Distribution*

The relation between tensor product binding units and Hebbian weights suggests another development of the present analysis. In McClelland's [20] Connection Information Distribution (CID) scheme, the activity of certain units determine the weights between others. Unbinding could be naturally carried out in a CID as follows. The represented structure would be active in a set of binder units which would set the weights between role and filler units. This would create a machine that transforms role patterns to filler patterns (to the approximation to which unbinding vectors equal role vectors). Figure 11 can be viewed as a CID in which the binder units are setting weights in a collection of $N$ role/filler associators.

Despite the intimate relation between tensor product binding units and connection weights, it should be emphasized that the primary purpose of the tensor product representation is not to serve as an apparatus for filler/role associations: it is rather to provide a pattern of activity representing a structured object which can then be used for processing the object as a whole. This is the reason why the elements of the tensor product representation have been viewed as the activities of units rather than the strength of connections. The CID allows us to use unit activities as connection strengths, giving us simultaneous access to both aspects of the representation.

### 3.6. Values as variables

It is often important for the value bound to a variable to in fact itself be a variable to which a value is to be bound. The tensor product binding representation allows for this in the following way. Out of the representation for the variable/value binding can be extracted the pattern of activity that represents the value. This pattern can in turn be used as the pattern representing a variable, and used in another binding on other binding units where it is bound to a value. The situation is depicted in Fig. 12.

### 3.7. Representation of symbolic operations; recursive decompositions

So far we have not considered the representation of symbolic *operations*: mappings from $S$ to itself. Examples that will now be considered are the stack operations *push* and *pop* and the LISP operators *car*, *cdr*, and *cons*. Understanding such operations are important for treating recursive role decomposi-

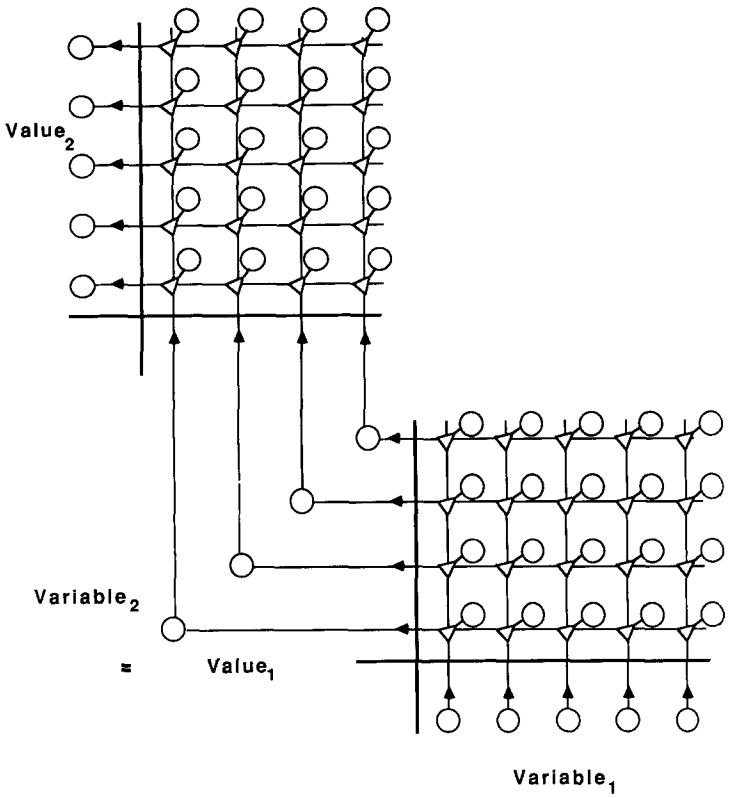


Fig. 12. A network capable of representing two value/variable bindings in which the same entity—the pattern of activity over the diagonally-aligned units—serves as the value in the first binding ($Value_1$) and the variable in the second binding ($Variable_2$).

tions, since in such a decomposition each role is in fact an operator mapping $S$ into $S$.

The definition we need to get started is

**Definition 3.7.** Let $O : S \rightarrow S$ be an operator on $S$ and $\Psi : S \rightarrow V$ be a connectionist representation of $S$. Then a corresponding *representation of* $O$ is an operator

$$\mathbf{O} : V \rightarrow V\ ; \qquad \mathbf{v} \mapsto \mathbf{Ov}$$

with the property

$$\Psi(O(s)) = \mathbf{O}\Psi(s)\ .$$

3.7.1. *Stack operations: push and pop*

In this section we consider the basic stack operations, *push* and *pop*. To keep complications to a minimum, two simplifications will be made. In place of a stack containing complex elements, simple strings from a fixed alphabet will be used to model the essential stack structure of linear ordered elements with a first element. The second simplification will be to consider an infinite stack, i.e., no limit to the length of the strings modeling the stack.

Let $S$ be the set of finite-length strings from an alphabet $A$. Let $F/R$ be the positional role decomposition of Definition 2.11. Let $\Psi_F$ be a faithful representation of $F$, and let $\Psi_R$ be a representation of $R$ in which the role vectors $\{\mathbf{r}_i\}_{i=0}^{\infty}$ representing the positional roles $\{r_i\}_{i=0}^{\infty}$ are all linearly independent. This means that $V_R$ is an infinite-dimensional space. (The analysis can easily be modified to strings of length no more than $n$, in which case $V_R$ can be $n$-dimensional; the finite case just introduces uninteresting complications.) For simplicity, assume that the role vectors span the space $V_R$ and therefore form a basis.

The positional role decomposition has the property that if $r_i$ is unbound, so is $r_j$ if $j > i$. Thus the representations of strings are all in a restricted subspace of $V$:

**Definition 3.8.** The *string subset* of $V$ is

$$V_S = \left\{ \sum_i \mathbf{f}_i \otimes \mathbf{r}_i \,\middle|\, \text{for all } i, \text{ if } \mathbf{f}_i = \mathbf{0}, \text{ then for all } j > i, \mathbf{f}_j = \mathbf{0} \right\}.$$

**Theorem 3.9.** *The pop operation on S is represented by a linear transformation* **pop** *on* $V$:

$$\mathbf{pop}: V \to V\ ; \qquad \sum_i \mathbf{f}_i \otimes \mathbf{r}_i \mapsto \sum_i \mathbf{f}_i \otimes \mathbf{r}_{i-1}\ .$$

*The operation* $push_a$ *on S is represented by an affine transformation* $\mathbf{push}_\mathbf{a}$ *on* $V$:

$$\mathbf{push}_\mathbf{a}: V \to V\ ; \qquad \sum_i \mathbf{f}_i \otimes \mathbf{r}_i \mapsto \mathbf{a} \otimes \mathbf{r}_0 + \sum_i \mathbf{f}_i \otimes \mathbf{r}_{i+1}\ .$$

*Both* **pop** *and* $\mathbf{push}_\mathbf{a}$ *map* $V_S$ *into* $V_S$, *for all* $\mathbf{a} \neq \mathbf{0}$.

**Proof.** First note that the definitions of **pop** and **push** given in the theorem are adequate because every vector in $V = V_F \otimes V_R$ can be uniquely expressed in the form $\Sigma_i \mathbf{f}_i \otimes \mathbf{r}_i$ since $\{\mathbf{r}_i\}$ is a basis of $V_R$. That **pop** is linear and **push** is affine is easily checked.

Suppose $s$ is the string $a_0 a_1 \ldots a_n$ and that the characters have representa-

tions $\Psi_F(a_i) = \mathbf{a}_i$. Then

$$\Psi(pop(s)) = \Psi(a_1 a_2 \ldots a_n)$$
$$= \sum_{i=1}^{n} \mathbf{a}_i \otimes \mathbf{r}_{i-1} = \mathbf{pop} \sum_{i=0}^{n} \mathbf{a}_i \otimes \mathbf{r}_i = \mathbf{pop}\Psi(s) .$$

Thus **pop** is a representation of *pop*. Similarly, $\mathbf{push}_\mathbf{a}$ is a representation of $push_a$.

$$\Psi(push_a(s)) = \Psi(a a_0 a_1 \ldots a_n)$$
$$= \mathbf{a} \otimes \mathbf{r}_0 + \sum_{i=0}^{n} \mathbf{a}_i \otimes \mathbf{r}_{i+1}$$
$$= \mathbf{push}_\mathbf{a} \sum_{i=0}^{n} \mathbf{a}_i \otimes \mathbf{r}_i = \mathbf{push}_\mathbf{a} \Psi(s) . \qquad \square$$

3.7.2. *LISP binary tree operations: car, cdr, and cons*

Let $S$ be the set of LISP S-expressions built from a set of atoms $A$. We define a role decomposition as follows. For the fillers, take $F = A$. A typical role, $r_{011011}$, is defined as follows. The predicate $a/r_{011011}$ is "the *caddaddr* is the atom $a$." The roles are indexed by finite bit strings, and correspond to compositions of *car* and *cdr* operations, with 0 indicating *car* and 1 indicating *cdr*. Note that these roles are to be filled only by *atoms*. Thus, for example, the $S$-expression $s = (a.(b.c))$ contains the bindings $\{a/r_0, b/r_{01}, c/r_{11}\}$; the role $r_1$ is unbound—not because $s$ has no *cdr*, but because the *cdr* is not an atom. The role indexed by the empty sting $\varepsilon$ is special: the predicate $a/r_\varepsilon$ is "is the atom $a$."

This decomposition is faithful and has single-valued roles. If objects like circular lists are considered valid $S$-expressions, then the decomposition is not finite.

This role decomposition has the property that if $r_x$ is bound, then $r_{yx}$ is unbound, where $yx$ is the concatenation of the bit strings $y$ and $x$. In particular, if $r_\varepsilon$ is bound, no other role can be; this is exactly the case for atoms. Lists are $S$-expressions for which the *cdr* is never a non-nil atom, at all levels of imbedding; in other words, for all bit strings $x$, $r_{1x}$ is unbound or bound to *nil*.

Let $\Psi_R$ map each $r_x$ into a corresponding vector $\mathbf{r}_x$ in a basis of an infinite-dimensional vector space $V_R$. Let $\Psi_F$ be a faithful representation of $F = A$ in $V_F$, and let $\mathbf{nil} := \Psi_F(nil)$. Now we investigate the properties of the induced tensor product representation $\Psi$.

**Definition 3.10.** The *atomic subspace* of $V$ is

$$V_a = \{\mathbf{f} \otimes \mathbf{r}_\varepsilon \mid \mathbf{f} \in V_F\} = V_F \otimes \mathrm{span}(\{\mathbf{r}_\varepsilon\}) .$$

The *non-atomic subspace* of $V$ is

$$V_{na} = \left\{ \sum_{x \neq \varepsilon} \mathbf{f}_x \otimes \mathbf{r}_x \,|\, \mathbf{f}_x \in V_F \right\} = V_F \otimes \text{span}(\{\mathbf{r}_x \,|\, x \neq \varepsilon\}) .$$

The *S-subset* of $V$ is

$$V_S = \left\{ \sum_x \mathbf{f}_x \otimes \mathbf{r}_x \,|\, \text{for all } x, \text{ if } \mathbf{f}_x = \mathbf{0}, \text{ then for all } y, \mathbf{f}_{yx} = \mathbf{0} \right\} .$$

The *list subset* of $V$ is

$$V_l = \left\{ \sum_x \mathbf{f}_x \otimes \mathbf{r}_x \in V_S \,|\, \text{for all } x, \mathbf{f}_{1x} \neq \mathbf{0} \Rightarrow \mathbf{f}_{1x} = \mathbf{nil} \right\} .$$

Note that $V_S$ is not closed under vector addition. For example, $\Psi((a)) + \Psi(((b)))$ corresponds to a mixture of two list structures; it possesses the bindings $\{a/r_0, b/r_{00}\}$, violating the condition defining $V_S$. Thus $V_S$ is not a vector space. The same example also shows that $V_l$ is not a vector space.

Now we are ready for representations of the operators *car*, *cdr*, and *cons*.

**Definition 3.11.** Define two linear transformations $\mathbf{T}_0$ and $\mathbf{T}_1$ on $V_R$ by the following actions on the basis $\{\mathbf{r}_x\}$:

$$\mathbf{T}_0 : V_R \to V_R ; \qquad \mathbf{r}_{x0} \mapsto \mathbf{r}_x ; \quad \mathbf{r}_{x1} \mapsto \mathbf{0} ; \quad \mathbf{r}_\varepsilon \mapsto \mathbf{0}$$

$$\mathbf{T}_1 : V_R \to V_R ; \qquad \mathbf{r}_{x1} \mapsto \mathbf{r}_x ; \quad \mathbf{r}_{x0} \mapsto \mathbf{0} ; \quad \mathbf{r}_\varepsilon \mapsto \mathbf{0}$$

**Theorem 3.12.** *The following linear transformations on V are representations of the operators car and cdr*:

$$\mathbf{car} : \sum_x \mathbf{f}_x \otimes \mathbf{r}_x \mapsto \sum_x \mathbf{f}_x \otimes \mathbf{T}_0 \mathbf{r}_x , \qquad \mathbf{cdr} : \sum_x \mathbf{f}_x \otimes \mathbf{r}_x \mapsto \sum_x \mathbf{f}_x \otimes \mathbf{T}_1 \mathbf{r}_x .$$

**Proof.** The representation of an $S$-expression $s$ can be written

$$\Psi(s) = \sum_y \mathbf{f}_y \otimes \mathbf{r}_y = \mathbf{f}_\varepsilon \otimes \mathbf{r}_\varepsilon + \sum_x \mathbf{f}_{x0} \otimes \mathbf{r}_{x0} + \sum_x \mathbf{f}_{x1} \otimes \mathbf{r}_{x1} .$$

Now $f_{x0} = cxr(car(s))$, where *cxr* denotes the composition of *car*s and *cdr*s corresponding to the bit string $x$. So if $t = car(s)$, then $f_{x0} = cxr(t)$. Thus the filler of $r_{x0}$ in $s$ is the filler of $r_x$ in $t = car(s)$. Conversely, any filler of $r_x$ in $t$ is a filler of $r_{x0}$ in $s$. Thus the representation of $car(s)$ is

$$\begin{aligned} \Psi(car(s)) &= \sum_x \mathbf{f}_{x0} \otimes \mathbf{r}_x \\ &= \mathbf{car}\left[ \mathbf{f}_\varepsilon \otimes \mathbf{r}_\varepsilon + \sum_x \mathbf{f}_{x0} \otimes \mathbf{r}_{x0} + \sum_x \mathbf{f}_{x1} \otimes \mathbf{r}_{x1} \right] = \mathbf{car}\Psi(s) . \end{aligned}$$

This shows that **car** represents *car*. By replaying this argument with *car* replaced by *cdr* and with 0 and 1 interchanged, we see that **cdr** represents *cdr*. The linearity of **car** and **cdr** are immediate consequences of the linearity of $\mathbf{T}_0$ and $\mathbf{T}_1$. □

**Remark**. The operators **car** and **cdr** treat **nil** like all other atoms: they map it to **0**. This corresponds to the *car* and *cdr* of all atoms, including *nil*, being *undefined*. If *car* and *cdr* are defined to be *undefined* on all non-nil atoms, but to take *nil* to *nil*, then the above definitions of **car** and **cdr** have to be changed if they are to represent *car* and *cdr*: the definitions must include the *ad hoc* stipulation that $\mathbf{nil} \otimes \mathbf{r}_\varepsilon$ is mapped to itself, while $\mathbf{a} \otimes \mathbf{r}_\varepsilon$ is mapped to **0** for all vectors **a** representing non-nil atoms. This does not destroy the linearity of **car** and **cdr** as long as the vector **nil** is linearly independent of the representations of all non-nil atoms. It does however destroy the property that $\mathbf{f} \otimes \mathbf{r} \mapsto \mathbf{f} \otimes \mathbf{Tr}$, where the transformation of the role is independent of its filler.

**Theorem 3.13.** *Let* $\boldsymbol{u}_0$ *and* $\boldsymbol{u}_1$ *be two vectors in V. Then there is a unique vector* **v** *in* $V_{na}$ *such that*

$$\mathbf{car}\ \mathbf{v} = \mathbf{u}_0 ,$$

$$\mathbf{cdr}\ \mathbf{v} = \mathbf{u}_1 .$$

*Define*

$$\mathbf{cons} : V \times V \to V_{na} ; \qquad (\mathbf{u}_0, \mathbf{u}_1) \mapsto \mathbf{v} .$$

*Then this function is*:

$$\mathbf{cons} : \left( \sum_x \mathbf{f}_x \otimes \mathbf{r}_x , \sum_y \mathbf{f}'_y \otimes \mathbf{r}_y \right) \mapsto \sum_x \mathbf{f}_x \otimes \mathbf{r}_{x0} + \sum_y \mathbf{f}'_y \otimes \mathbf{r}_{y1} .$$

**cons** *is a representation of the cons function on S.*

**Proof.** Let

$$\mathbf{u}_0 = \sum_x \mathbf{f}_x \otimes \mathbf{r}_x ,$$

$$\mathbf{u}_1 = \sum_y \mathbf{f}'_y \otimes \mathbf{r}_y ,$$

$$\mathbf{v} = \sum_x \mathbf{f}_x \otimes \mathbf{r}_{x0} + \sum_y \mathbf{f}'_y \otimes \mathbf{r}_{y1} .$$

Then

$$\mathbf{car}\,\mathbf{v} = \mathbf{car}\left[\sum_x \mathbf{f}_x \otimes \mathbf{r}_{x0} + \sum_y \mathbf{f}'_y \otimes \mathbf{r}_{y1}\right] = \sum_x \mathbf{f}_x \otimes \mathbf{r}_x = \mathbf{u}_0$$

and

$$\mathbf{cdr}\,\mathbf{v} = \mathbf{cdr}\left[\sum_x \mathbf{f}_x \otimes \mathbf{r}_{x0} + \sum_y \mathbf{f}'_y \otimes \mathbf{r}_{y1}\right] = \sum_y \mathbf{f}'_y \otimes \mathbf{r}_y = \mathbf{u}_1 .$$

Furthermore, $\mathbf{v} \in V_{na}$ so $\mathbf{v}$ satisfies the required conditions. These conditions completely determine $\mathbf{v}$: the **car** condition determines the fillers of all $\{r_{x0}\}$, the **cdr** condition determines the fillers of all $\{r_{x1}\}$, and the condition that $\mathbf{v}$ be in $V_{na}$ implies that the only remaining role, $r_\varepsilon$, must be unfilled.

Since **car** and **cdr** represent *car* and *cdr*, it follows that **cons** represents *cons*. To see this, let

$$s = cons(s_0, s_1) ,$$

$$\mathbf{u}_0 = \Psi(s_0) ,$$

$$\mathbf{u}_1 = \Psi(s_1) .$$

Then, since **car** represents *car*, and $car(s) = s_0$ ,

$$\mathbf{car}\,\Psi(s) = \Psi(car(s)) = \Psi(s_0) = \mathbf{u}_0$$

and similarly

$$\mathbf{cdr}\,\Psi(s) = \mathbf{u}_1 .$$

By the previous part of the proof, this implies that

$$\Psi(s) = \mathbf{cons}(\mathbf{u}_0, \mathbf{u}_1) .$$

In other words,

$$\Psi(cons(s_0, s_1)) = \mathbf{cons}(\Psi(s_0), \Psi(s_1)) .$$

Thus **cons** represents *cons*. □

Just as complex structures in $S$ can be constructed from atoms by successive applications of *cons*, so the tensor product representation of these items can similarly be constructed by successive applications of **cons** on the vectors

representing atoms:

$$\Psi(a) = \Psi_F(a) \otimes \mathbf{r}_\varepsilon .$$

Using **cons** to build up complex representations from simpler ones allows us to exploit the recursive role decomposition of $S$ provided by *car* and *cdr*.

The analysis of strings in Section 3.7.1 can be viewed as a subset of this analysis of $S$-expressions. The alphabet is identified with the set of atoms, and the $i$th positional role $r_i$ of the string is identified with $r_{0i_u}$, where $i_u$ is the unary representation of $i$: $i_u = 11\ldots1$ ($i$ times). The operator *pop* becomes *cdr* and $push_a(s)$ becomes $cons(a, s)$.

As emphasized in the introduction, the primary goal of this research is representations of structured data that can support effective massively parallel processing. As an example, with the representation of binary trees discussed here, it is possible to determine in one operation whether the atom $a$ appears anywhere in the tree; as pointed out in Section 3.1, the unbinding operations can be performed with respect to fillers as well as roles, and with a simple linear unbinding operation we can compute what role is filled by $a$. (As mentioned in Section 3.1, if there are multiple roles filled by $a$, the pattern computed will be the superposition of the vectors representing those roles).[7]

### 3.7.3. *Recursive construction of tensor product representations*

Related to recursive decomposition is the recursive construction of tensor product representations. This occurs when the fillers or roles are themselves structures that are decomposed by a new role decomposition. In other words, having decomposed $S$ in terms of $F$ and $R$, we now take $F$ or $R$ as a new $S'$ and decompose it in terms of new fillers $F'$ and roles $R'$. Consider the case of decomposition of $R$. If the role decomposition of $R$ is $F'/R'$, then the binding $f/r$ is itself a set of bindings $f/(f'/r')$. The tensor product representation of such a finer-grained binding is then

$$\mathbf{f} \otimes (\mathbf{f}' \otimes \mathbf{r}') .$$

In this case we are led to third-order (or, by further recursion, higher-order) tensor products, that is, to tensors of rank three or higher. The binding units can be interpreted as representing third- (or higher-) order conjunctions of features.

This recursive structure is just what we see in the Rumelhart and McClelland [33] past-tense learning model. Here the original role decomposition of phonetic strings is the 1-neighbor context decomposition. Each role $r_{x\_y}$ is itself

[7] Determining whether a tree contains a certain subtree (rather than a certain atom) can also be done in a single operation, if the role vectors are chosen in an appropriately recursive fashion; however this requires further development of the analysis that goes beyond the scope of this paper.

a structured object, whose structure is determined by the pair $(x, y)$. These pairs can be decomposed by the right-neighbor role decomposition, in which $x$ fills the role *has right neighbor* $y$, $r'_{\_y}$. Thus the binding $i/r_{w\_d}$ (the vowel in *weed*) becomes $i/(w/r'_{\_d})$ and the final representation is the third-order tensor product

$$\mathbf{i} \otimes \mathbf{w} \otimes \mathbf{r}'_{\_d} .$$

In fact, in this model, this is just $\mathbf{i} \otimes \mathbf{w} \otimes \mathbf{d}$, since the role vector $\mathbf{r}'_{\_d}$ is just $\mathbf{d}$. Equivalently, we can take the more naturally ordered product $\mathbf{w} \otimes \mathbf{i} \otimes \mathbf{d}$ as the representation of the subsequence *wid*. (This same result could have been arrived at through other routes, e.g., a left-neighbor decomposition of the 1-neighbor contextual roles.)

At this point we can now consider whether it would not have been better to view a structure not as a set of roles and fillers, but rather as a set of constituents engaged in certain mutual *relations*. The letter sequence *abc*, e.g., could be viewed as the constituents $a$, $b$, and $c$ engaged in the relations *left-of*$(a, b)$ and *left-of*$(b, c)$. This could be used to construct a tensor product representation in which *left-of*$(a, b)$ was represented by the tensor product of three vectors representing *left-of*, $a$ and $b$.

This variation of the role/filler construction adopted above has certain advantages, but in the end produces a representation which is just a special case of the role/filler construction. As we have just seen, using contextual roles, we can view *abc* as having roles *left_of_b*, *left_of_c* filled, respectively, by $a$ and $b$. Then, considering the roles in turn as structures, we view *left_of_b* as having a subrole *left_of* filled by $b$. Then the recursive role/filler construction leads to a representation of $a/left\_of\_b$ which is the tensor product of the three vectors representing *left_of*, $b$ and $a$; this representation is equivalent to that of *left_of*$(a, b)$.

Essentially, applying the role/filler construction recursively using contextual roles amounts to a standard trick from mathematical logic for reducing functions and relations taking multiple arguments—e.g., the two-place relation *left_of*—to nested functions and relations taking only one argument—e.g., the one-place relation *left_of_b* constructed from the one-place function *left_of*: *left_of*$(a, b) =$ *left_of_b*$(a)$; *left_of_b* = *left_of*$(b)$. In this sense, structural decomposition via multiple-argument functions and relations can be seen as syntactic sugar for certain kinds of recursive filler/role decompositions.

### 3.8. Storage of structured data in connectionist memories

One of the primary uses of connectionist representations is as objects of associations in associative memories: associative memories are connectionist-implemented mappings from a vector representing a cue to a vector representing the retrieved item. Because of its mathematical simplicity it is possible to

analyze the use of tensor product representations in such memories. Here I analyze the case of pair association since it is simpler than the content-addressed auto-association case which is perhaps a purer example of connectionist "memory" [30].

We start with the simplest possible case.

**Theorem 3.14.** *Suppose $\Psi_{F/R}$ is a tensor product representation of S induced by a decomposition with single-valued roles, with representations of fillers and roles in which all filler vectors are mutually orthogonal as are all role vectors. Let $\{s^{(k)}\}$ be a subset of S, and let the vectors representing these structures, $\{\mathbf{s}^{(k)}\}$, be associated in a connectionist network using the Hebb rule with the patterns $\{\mathbf{t}^{(k)}\}$. Then if the structures $\{s^{(k)}\}$ share no common fillers (i.e., for each role, all structures have different fillers), the associator will function perfectly; otherwise there will be cross-talk that is monotonic in the degree of shared fillers. In particular, the output associated with $s^{(l)}$ is proportional to*

$$\mathbf{t}^{(l)} + \sum_{k \neq l} \mu_{lk} \mathbf{t}^{(k)}$$

*where*

$$\mu_{lk} = \frac{\sum_{i : f_i^{(l)} = f_i^{(k)}} \|\mathbf{f}_i^{(l)}\|^2 \|\mathbf{r}_i\|^2}{\sum_i \|\mathbf{f}_i^{(l)}\|^2 \|\mathbf{r}_i\|^2} .$$

**Proof.** The Hebbian weights are

$$\mathbf{W} = \sum_k \mathbf{t}^{(k)} \mathbf{s}^{(k)\mathrm{T}} .$$

Thus the output generated from the input representing $s^{(l)}$ is:

$$\begin{aligned}
\mathbf{W}\mathbf{s}^{(l)} &= \sum_k \mathbf{t}^{(k)} \mathbf{s}^{(k)\mathrm{T}} \mathbf{s}^{(l)} \\
&= \sum_k \mathbf{t}^{(k)} \left[ \sum_i \mathbf{f}^{(k)} \otimes \mathbf{r}_i \right] \cdot \left[ \sum_j \mathbf{f}_j^{(l)} \otimes \mathbf{r}_j \right] \\
&= \sum_k \mathbf{t}^{(k)} \sum_i \sum_j (\mathbf{f}_i^{(k)} \cdot \mathbf{f}_j^{(l)})(\mathbf{r}_i \cdot \mathbf{r}_j) \\
&= \sum_k \mathbf{t}^{(k)} \sum_i \sum_j (\delta_{\mathbf{f}_i^{(k)}, \mathbf{f}_j^{(l)}} \|\mathbf{f}_i^{(k)}\|^2)(\delta_{ij} \|\mathbf{r}_i\|^2) \\
&= \sum_k \mathbf{t}^{(k)} \sum_i \delta_{\mathbf{f}_i^{(k)}, \mathbf{f}_j^{(l)}} \|\mathbf{f}_i^{(k)}\|^2 \|\mathbf{r}_i\|^2 \\
&= \left[ \sum_i \|\mathbf{f}_i^{(k)}\|^2 \|\mathbf{r}_i\|^2 \right] \mathbf{t}^{(l)} + \sum_{k \neq l} \left[ \sum_i \|\mathbf{f}_i^{(l)}\|^2 \|\mathbf{r}_i\|^2 \delta_{\mathbf{f}_i^{(k)}, \mathbf{f}_i^{(l)}} \right] \mathbf{t}^{(k)} .
\end{aligned}$$

The first term here is the correct associate $\mathbf{t}^{(l)}$ weighted by a positive coefficient. The second term is a sum of all other (incorrect) associates $\{\mathbf{t}^{(k)}\}_{k \neq l}$, each weighted by a nonnegative coefficient. These coefficients will all vanish if there are no common fillers. Taking the ratio of the coefficient of $\mathbf{t}^{(k)}$ to that of $\mathbf{t}^{(l)}$ gives the desired result. □

The Hebb rule is capable of accurately learning associations to patterns that are orthogonal. If the patterns are not necessarily orthogonal but are still linearly independent, the associations can be accurately stored in a connectionist memory using the more complex Widrow–Hoff [49] or delta learning procedure [30]. So the question is, what collections of symbolic structures have linearly independent representations under the tensor product representation? To answer this question, it turns out to be important to define the following concept:

**Definition 3.15.** Let $F/R$ be a role decomposition of $S$ and let $k \mapsto s^{(k)}$ be a sequence of elements in $S$. An *annihilator* of $k \mapsto s^{(k)}$ with respect to $R/F$ is a sequence of real numbers $k \mapsto \alpha^{(k)}$, not all zero, such that, for all fillers $f \in F$, and all roles $r \in R$,

$$\sum_{k\,:\,f/r \in \beta(s^{(k)})} \alpha^{(k)} = 0\,.$$

For example, consider the sequence of strings $(ax, bx, ay, by)$. With respect to the positional role decomposition, this has annihilator $(+1, -1, -1, +1)$, since for each filler/role binding in $\{a/r_1, b/r_1, x/r_2, y/r_2\}$, the corresponding annihiulator elements are $\{+1, -1\}$, which sum to zero.

**Theorem 3.16.** *Suppose $\Psi$ is a tensor product representation of the structures $S$, and that $k \mapsto s^{(k)}$ is a sequence of distinct elements in $S$. Suppose that the filler vectors* $\mathbf{f}$ *representing the fillers bound in the elements $\{s^{(k)}\}$ are all linearly independent, and that the same is true of the role vectors* $\mathbf{r}$ *representing the roles bound in the elements $s^{(k)}$. If $k \mapsto s^{(k)}$ has no annihilator with respect to $F/R$, then associations to the tensor product representations $\{\Psi(s_i)\}$ can all be simultaneously and accurately stored in a connectionist memory by using the Widrow–Hoff learning rule.*

**Proof.** Let

$$\Psi(s^{(k)}) = \sum_i \mathbf{f}_i^{(k)} \otimes \mathbf{r}_i\,.$$

Here we use the same set of roles $\{\mathbf{r}_i\}$ for all structures $\{s^{(k)}\}$; this can always be done provided we allow the filler vector $\mathbf{f}_i^{(k)}$ to equal the zero vector whenever the role $\mathbf{r}_i$ is unbound in structure $s^{(k)}$.

By the remarks immediately preceding Definition 3.15, it is sufficient to show that the patterns $\{\Psi(s^{(k)})\}$ are all linearly independent. Suppose on the contrary that there are coefficients $\{\alpha^{(k)}\}$, not all zero, such that

$$\mathbf{0} = \sum_k \alpha^{(k)} \Psi(s^{(k)})$$
$$= \sum_k \alpha^{(k)} \left[ \sum_i \mathbf{f}_i^{(k)} \otimes \mathbf{r}_i \right] = \sum_i \left[ \sum_k \alpha^{(k)} \mathbf{f}_i^{(k)} \right] \otimes \mathbf{r}_i .$$

Then, because the role vectors $\{\mathbf{r}_i\}$ are linearly independent, this implies that for all $i$,

$$\sum_k \alpha^{(k)} \mathbf{f}_i^{(k)} = \mathbf{0} .$$

Now we rewrite this as a sum over all distinct filler vectors:

$$\sum_\gamma \mathbf{f}_\gamma \sum_{k\,:\,\mathbf{f}_i^{(k)} = \mathbf{f}_\gamma} \alpha^{(k)} = \mathbf{0} .$$

But since the filler vectors $\{\mathbf{f}_\gamma\}$ are linearly independent, this implies, for all $i$ and for all $\gamma$,

$$\sum_{k\,:\,\mathbf{f}_i^{(k)} = \mathbf{f}_\gamma} \alpha^{(k)} = 0 .$$

This means exactly that $\{\alpha^{(k)}\}$ is an annihilator of the sequence of structures $k \mapsto s^{(k)}$. Since by hypothesis such an annihilator does not exist, it must be that the representations $\{\Psi(s^{(k)})\}$ are linearly independent. □

It was remarked above that the strings $\{ax, bx, ay, by\}$ possess an annihilator with respect to the positional role decomposition. This means that the tensor product representations of these strings are not linearly independent, even under the preceding theorem's assumptions of linearly independent filler and role vectors. They cannot therefore be accurately associated with arbitrary patterns even using the Widrow–Hoff learning rule. On the other hand, it is easy to see that the strings $\{ax, bx, ay\}$ do *not* possess an annihilator; the preceding theorem shows that their tensor product representations can therefore be accurately associated with arbitrary patterns.

### 3.9. Learning optimal role representations

The tensor product representation is constructed from connectionist representations of fillers and roles. As indicated in Section 2.3.2.3, distributed

representation of fillers have been used in many connectionist models for some time; usually, these representations are built from an analysis of the fillers in terms of features relevant for the task being performed. But what about distributed representations of roles? This, a problem raised in [13], becomes a major question in tensor product representation. For many applications, it is easy to imagine task-appropriate features for roles that could serve well as the basis for distributed role representations. For example, Fig. 3 shows a distributed representation of positional roles with the useful property that nearby positions are represented by similar patterns. Algorithms such as back-propagation [31] can also be used to learn role representations for a given task, using a network such as that shown in Fig. 11 and backpropagating through the multiplicative junctions.

It is also possible to analyze the question of distributed representations for roles from a domain-independent perspective. One rather general sense in which a set of role vectors might be considered "optimal" can be characterized as those representations for which fillers can be unbound in a way that minimizes the total error introduced by non-linearly independent role vectors. In [39], I introduce this error measure, give algebraic and geometric characterizations of optimal sets of role vectors, and show how a simple recurrent linear network can perform gradient descent in the error measure to find the optimal vectors. This learning algorithm is an example of a "recirculation algorithm" [14] in which activity cycles in a loop, and the change in a weight $w_{ji}$ from $i$ to $j$ is proportional to the activity at $i$ times the *rate of change* of activity at $j$. Space does not allow presentation of this analysis here, and the reader is referred to [39] for full detail.

## 4. Conclusion

The limitations of the results reported here are many. The theoretical analyses of role decompositions, graceful saturation, connectionist representations of symbolic operators and recursive structures, retrieval of tensor product representations in connectionist memories, and optimal role vectors are just beginnings. An analysis is needed of the consequences of throwing away binding units and other means of controlling the potentially prohibitive growth in their number. A further analysis is needed of the possibility of having a value for one variable serve as another variable, without an unbinding of the first variable. The relations between tensor product binding units and connection weights, briefly considered in Section 3.5, need to be pursued. Furthermore, for reasons such as those discussed in [13], it will often happen that for storing representations of structures in content-addressed memories, the tensor product representation alone will be insufficient, and hidden units will need to be added to capture higher-order conjunctions that distinguish different structures from each other; the current framework needs to be extended to adequately

treat such cases. Also, the analysis of recursive structures needs to be further developed to ensure that imbedded structures can be effectively processed.

The viability of the tensor product representation has been confirmed in an implemented connectionist model, the tensor product production system (TPPS) [6, 7]. TPPS performs a number of fundamental aspects of symbolic processing, such as condition matching against symbolic structures, variable binding on the condition side and substitution into the action side of productions, conflict resolution, and insertion and deletion of structures. TPPS can be viewed as a reimplementation of Touretzky and Hinton's distributed connectionist production system (DCPS) [46], intended to show that a tensor-product-based system would be simpler from both mathematical and implementation perspectives, and would also perform well.[8]

In summary, the tensor product representation enables truly distributed representations of complex symbolic structures in connectionist systems, in a natural way that generalizes existing representations and is simple enough to permit analyses of a number of properties. Tensor product representations are determined by a number of parameters which can be productively analyzed separately: the role decomposition of the structures being represented, the method for connectionist representation of conjunction, and the connectionist representations of fillers and roles being used. Such conceptual tools for analyzing alternative connectionist representations are necessary if we are to deepen our understanding of the representational component of connectionist modeling. Most importantly, the tensor product framework allows a crucial aspect of symbolic computation, the representation and processing of structured data based on the binding of values to variables, to be incorporated into the connectionist approach in a natural way that adds to the power of connectionist computation without sacrificing its advantages.

ACKNOWLEDGEMENT

This paper attempts to formalize and analyze ideas on distributed representation that have been articulated and exploited in various ways by a number of connectionist researchers. I have benefitted in particular from many ideas of Geoff Hinton, both published and personally communicated. The responsibility for the formulation pursued here is of course entirely my own.

This work has been supported by NSF grants IRI-8609599 and ECE-8617947 to the author, by a

[8] In this paper, "variable binding" refers to the creation of an object that links a variable to its value; this can be applied to *implicit* bindings such as that between a letter and the position it occupies in a string, or to *explicit* bindings such as that between a symbol denoting a variable in the condition or action side of a production and a symbol denoting the value of that variable. In implicit bindings, the "variable" (or "role") involved is not usually explicitly represented in a symbolic system by a symbol, but rather is implicit in the datastructures used in the implementation. This needn't be the case, of course; instead of writing a list as (*a b c*), we *could* use a frame-like structure such as *list*: *el_1* = *a*, *el_2* = *b*, *el_3* = *c*; then the implicit binding of the filler *a* to its role would become explicit. Explicit variable binding was the primary issue in DCPS and later TPPS.

grant to the author from the Sloan Foundation's computational neuroscience program, and by the Optical Connectionist Machine Program of the NSF Engineering Research Center for Optoelectronic Computing Systems at the University of Colorado at Boulder, supported in large part by NSF grant CDR 8622236.

Many thanks to Géraldine Legendre for crucial support.

## REFERENCES

1. J.A. Anderson and G.E. Hinton, Models of information processing in the brain, in: G.E. Hinton and J.A. Anderson, eds., *Parallel Models of Associative Memory* (Erlbaum, Hillsdale, NJ, 1981).
2. M. Derthick, A connectionist architecture for representing and reasoning about structured knowledge, in: *Proceedings Ninth Annual Conference of the Cognitive Science Society*, Seattle, WA (1987).
3. C.P. Dolan, Tensor manipulation networks: Connectionist and symbolic approaches to comprehension, learning, and planning, AI Lab Tech. Rept., University of California, Los Angeles, CA (1989).
4. C.P. Dolan and M.G. Dyer, Symbolic schemata, role binding, and the evolution of structure in connectionist memories, in: *Proceedings First International Conference on Neural Networks*, San Diego, CA (1987).
5. C.P. Dolan and M.G. Dyer, Parallel retrieval of conceptual knowledge, in: D. Touretzky, G.E. Hinton and T.J. Sejnowski, eds., *Proceedings Connectionist Models Summer School* (Morgan Kaufmann, Los Altos, CA, 1988).
6. C.P. Dolan and P. Smolensky, Implementing a connectionist production system using tensor products, in: D.Touretzky, G.E. Hinton and T.J. Sejnowski, eds., *Proceedings Connectionist Models Summer School* (Morgan Kaufmann, Los Altos, CA, 1988).
7. C.P. Dolan and P. Smolensky, Tensor product production system: A modular architecture and representation, *Connection Sci.* **1** (1989) 53–68.
8. M. Fanty, Context-free parsing in connectionist networks, Tech. Rept. 174, Department of Computer Science, University of Rochester, Rochester, NY (1985).
9. J.A. Feldman, Four frames suffice: A provisional model of vision and space, *Behav. Brain Sci.* **8** (1985) 265–289.
10. J.A. Feldman, Neural representation of conceptual knowledge, Tech. Rept. 189, Department of Computer Science, University of Rochester, Rochester, NY (1986).
11. J.A. Fodor and Z.W. Pylyshyn, Connectionism and cognitive architecture: A critical analysis, *Cognition* **28** (1988) 3–71.
12. G.E. Hinton, A parallel computation that assigns canonical object-based frames of reference, in: *Proceedings IJCAI-81*, Vancouver, BC (1981).
13. G.E. Hinton, Implementing semantic networks in parallel hardware, in: G.E. Hinton and J.A. Anderson, eds., *Parallel Models of Associative Memory* (Erlbaum, Hillsdale, NJ, 1981).
14. G.E. Hinton and J.L. McClelland, Learning representations by recirculation, in: D.Z. Anderson, ed., *Neural Information Processing Systems* (American Institute of Physics, New York, 1988).
15. G.E. Hinton, J.L. McClelland and D.E. Rumelhart, Distributed representations, in: J.L. McClelland, D.E. Rumelhart and the PDP Research Group, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* **2**: *Psychological and Biological Models* (MIT Press/Bradford Books, Cambridge, MA, 1986).
16. M.I. Jordan, An introduction to linear algebra in parallel distributed processing, in: D.E. Rumelhart, J.L. McClelland and the PDP Research Group, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* **1**: *Foundations* (MIT Press/Bradford Books, Cambridge, MA, 1986).

17. J. Lachter and T.G. Bever, The relation between linguistic structure and associative theories of language learning: A constructive critique of some connectionist learning models, *Cognition* **28** (1988) 195–247.
18. Y. Le Cun, C.C. Galland and G.E. Hinton, GEMINI: Gradient estimation through matrix inversion after noise injection, in: D.S. Touretzky, ed., *Advances in Neural Information Processing Systems* **1** (Morgan Kaufmann, San Mateo, CA, 1989).
19. L.H. Loomis and S. Sternberg, *Advanced Calculus* (Addison-Wesley, Reading, MA, 1968) 305–320.
20. J.L. McClelland, The programmable blackboard model of reading, in: J.L. McClelland, D.E. Rumelhart and the PDP Research Group, eds., *Parallel Distributed Processing*: *Explorations in the Microstructure of Cognition* **2**: *Psychological and Biological Models* (MIT Press/ Bradford Books, Cambridge, MA, 1986).
21. J.L. McClelland and J.L. Elman, Interactive processes in speech perception: The TRACE model, in: J.L. McClelland, D.E. Rumelhart and the PDP Research Group, eds., *Parallel Distributed Processing*: *Explorations in the Microstructure of Cognition* **2**: *Psychological and Biological Models* (MIT Press/Bradford Books, Cambridge, MA, 1986).
22. J.L. McClelland and A.H. Kawamoto, Mechanisms of sentence processing: Assigning roles to constituents, in: J.L. McClelland, D.E. Rumelhart and the PDP Research Group, eds., *Parallel Distributed Processing*: *Explorations in the Microstructure of Cognition* **2**: *Psychological and Biological Models* (MIT Press/Bradford Books, Cambridge, MA, 1986).
23. J.L. McClelland and D.E. Rumelhart, An interactive activation model of context effects in letter perception, Part 1: An account of the basic findings, *Psychol. Rev.* **88** (1981) 375–407.
24. J.L. McClelland, D.E. Rumelhart and the PDP Research Group, eds., *Parallel Distributed Processing*: *Explorations in the Microstructure of Cognition* **2**: *Psychological and Biological Models* (MIT Press/Bradford Books, Cambridge, MA, 1986).
25. M.C. Mozer, *The Perception of Multiple Objects*: *A Parallel, Distributed Processing Approach* (MIT Press/Bradford Books, Cambridge, MA, 1990).
26. E. Nelson, *Tensor Analysis* (Princeton University Press, Princeton, NJ, 1967).
27. S. Pinker and A. Prince, On language and connectionism: Analysis of a parallel distributed processing model of language acquisition, *Cognition* **28** (1988) 73–193.
28. A. Prince and S. Pinker, Wickelphone ambiguity, *Cognition* **30** (1988) 189–190.
29. M.S. Riley and P. Smolensky, A parallel model of (sequential) problem solving, in: *Proceedings Sixth Annual Conference of the Cognitive Science Society*, Boulder CO (1984).
30. D.E. Rumelhart, G.E. Hinton and J.L. McClelland, A general framework for parallel distributed processing, in: D.E. Rumelhart, J.L. McClelland and the PDP Research Group, eds., *Parallel Distributed Processing*: *Explorations in the Microstructure of Cognition* **1**: *Foundations* (MIT Press/Bradford Books, Cambridge, MA, 1986).
31. D.E. Rumelhart, G.E. Hinton and R.J. Williams, Learning internal representations by error propagation, in: D.E. Rumelhart, J.L. McClelland and the PDP Research Group, eds., *Parallel Distributed Processing*: *Explorations in the Microstructure of Cognition* **1**: *Foundations* (MIT Press/Bradford Books, Cambridge, MA, 1986).
32. D.E. Rumelhart and J.L. McClelland, An interactive activation model of context effects in letter perception, Part 2: The contextual enhancement effect and some tests and extensions of the model, *Psychol. Rev.* **89** (1982) 60–94.
33. D.E. Rumelhart and J.L. McClelland, On learning the past tenses of English verbs, in: J.L. McClelland, D.E. Rumelhart and the PDP Research Group, eds., *Parallel Distributed Processing*: *Explorations in the Microstructure of Cognition* **2**: *Psychological and Biological Models* (MIT Press/Bradford Books, Cambridge, MA, 1986).
34. D.E. Rumelhart, J.L. McClelland and the PDP Research Group, eds., *Parallel Distributed Processing*: *Explorations in the Microstructure of Cognition* **1**: *Foundations* (MIT Press/ Bradford Books, Cambridge, MA, 1986).

35. T.J. Sejnowski and C.R. Rosenberg, Parallel networks that learn to pronounce English text, *Complex Syst.* **1** (1987) 145–168.
36. P. Smolensky, Information processing in dynamical systems: Foundations of harmony theory, in: D.E. Rumelhart, J.L. McClelland, and the PDP Research Group, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* **1**: *Foundations* (MIT Press/Bradford Books, Cambridge, MA, 1986).
37. P. Smolensky, Neural and conceptual interpretations of parallel distributed processing models, in: J.L. McClelland, D.E. Rumelhart and the PDP Research Group, eds., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* **2**: *Psychological and Biological Models* (MIT Press/Bradford Books, Cambridge, MA, 1986).
38. P. Smolensky, Connectionist AI, symbolic AI, and the brain, *AI Rev.* **1** (1987) 95–109. Special Issue on the Foundations of AI.
39. P. Smolensky, On variable binding and the representation of symbolic structures in connectionist systems, Tech. Rept. CU-CS-355-87, Department of Computer Science, University of Colorado at Boulder, CO (1987).
40. P. Smolensky, The constituent structure of connectionist mental states: A reply to Fodor and Pylyshyn, *Southern J. Philos.* **26**, Supplement (1987) 137–163.
41. P. Smolensky, On the proper treatment of connectionism, *Behav. Brain Sci.* **11** (1988) 1–23.
42. P. Smolensky, Connectionism, constituency, and the language of thought, in: B. Loewer and G. Rey, eds., *Fodor and His Critics* (Blackwell, Oxford, 1991).
43. P. Smolensky and M.C. Mozer, *Lectures on Connectionist Cognitive Modeling* (Erlbaum, Hillsdale, NJ, to appear).
44. D.S. Touretzky BoltzCONS: Reconciling connectionism with the recursive nature of stacks and trees, in: *Proceedings Eighth Annual Conference of the Cognitive Science Society*, Amherst, MA (1986).
45. D.S. Touretzky and S. Geva, A distributed connectionist representation for concept structures, in: *Proceedings Ninth Annual Conference of the Cognitive Science Society*, Seattle, WA (1987).
46. D.S. Touretzky and G.E. Hinton, Symbols among the neurons: Details of a connectionist inference architecture, in: *Proceedings IJCAI-85*, Los Angeles, CA (1985).
47. A.M. Treisman and H. Schmidt, Illusory conjunctions in the perception of objects, *Cognitive Psychol.* **14** (1982) 107–141.
48. F.W. Warner, *Foundations of Differentiable Manifolds and Lie Groups* (Scott, Foresman, Glenview, IL, 1971) 54–62.
49. G. Widrow and M.E. Hoff, Adaptive switching circuits, in: *Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record, Part 4* (1960) 96–104.
50. D. Willshaw, Holography, associative memory, and inductive generalization, in: G.E. Hinton and J.A. Anderson, eds., *Parallel Models of Associative Memory* (Erlbaum, Hillsdale, NJ, 1981).