TENSOR REPRESENTATIONS AND HARMONY THEORY: A CRITICAL ANALYSIS

by

René Gourley

B.Math University of Waterloo, 1992

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
June, 1995

ISBN 0-612-16893-X

Canada

APPROVAL

Name: René Gourley

Degree: Master of Science

Title of thesis: Tensor Representations and Harmony Theory: A Critical Analysis

Examining Committee: Ramesh Krishnamurti
Chair

Dr. Robert Hadley
Senior Supervisor
Associate Professor of Computing Science

Dr. Arvind Gupta
Senior Supervisor
Assistant Professor of Computing Science

Dr. James Delgrande
External Examiner
Associate Professor of Computing Science

Date Approved: June 8, 1995

SIMON FRASER UNIVERSITY

PARTIAL COPYRIGHT LICENSE

Title of Thesis/Project/Extended Essay

Tensor Representations and Harmony Theory: A Critical Analysis.

Author:

(signature)

René Gourley

(name)

July 11, 1995

(date)

Abstract

Harmony theory and tensor representations have been proposed as a means by which connectionist models can accept formal languages. Their proponents aim to provide a neural explanation of the productivity and systematicity of cognitive processes, without directly implementing symbolic algorithms. Via tensor representation, this theory interprets the activation vector of a connectionist system as a parse tree for a string in a particular context free language. Harmony theory apparently describes how to construct a network whose stable equilibria represent valid parse trees.

This thesis presents a detailed analysis of tensor representations and harmony theory. Over the course of this exposition, errors in the original formulation are identified and improvements are proposed.

The thesis then goes on to examine some major issues confronting harmony theory. The first issue is that of input and output which have not been satisfactorily defined by harmony theorists. Secondly, we examine the very large size of the networks. Finally, this thesis inspects harmony theory relative to its own goals and shows that the constructed networks admit stable equilibria that do not represent valid parse trees. Thus, harmony theory is unable to support its advocates' bold claims.

Contents

Abstract iii

1 Overture 1

1.1 The Challenge 1

1.2 The Answer 3

1.3 The Rebuttal: The Main Result 5

2 Background and Notation 7

2.1 Sets 7

2.2 Graph Theory 8

2.2.1 Definition 8

2.3 Formal Languages 10

2.3.1 Definition 10

2.3.2 Grammars 10

2.3.3 Derivation Graphs 11

2.3.4 Machines and Languages 12

2.4 Linear Algebra 14

2.5 Tensors 14

2.6 Neural Networks 17

3 Tensor Representations 20

3.1 Simple Tensor Representations 20

3.2 Recursive Tensor Representations 24

3.3 Representation of Parse Trees 29
3.4 Representation of Derivation Graphs 32
3.4.1 Normal Form for Type 0 Grammars 32
3.4.2 Spanning Tree Representation of Directed Acyclic Graphs . . 33
3.4.3 Recursive Representation of Directed Acyclic Graphs 36

4 Harmony **39**
4.1 Symbolic Formulation 40
4.1.1 Context-Free Parse Trees 40
4.1.2 Type 0 Derivation Graphs 44
4.1.3 The Symbolic Viewpoint 44
4.2 Numeric Formulation 45
4.2.1 Connectionist Networks as Energy Minimization Systems . . . 47
4.2.2 Pairwise Harmony in Context Free Parse Trees 51
4.2.3 Harmony of Whole Context Free Parse Trees 56

5 Discord **63**
5.1 Input and Output in Harmony Networks 63
5.1.1 Input 64
5.1.2 Output 66
5.1.3 Decision Networks 68
5.1.4 Harmonic Decision Networks 71
5.1.5 Symmetric Harmonic Decision Networks 72
5.2 Network Size 79
5.3 Harmony Networks Do Not Work 81
5.3.1 More Activations Than Trees 82
5.3.2 Non-Negative Eigenvalues Yield Non-Trees 83
5.3.3 Negative Eigenvalues Yields Non-Tree 86
5.3.4 Conclusion 88

6 Resolution **89**

List of Tables

2.1 Notation for Sets 8
2.2 Notation for Linear Algebra 15

3.1 Normal form for a Type 0 grammar 33

4.1 Rules for determining the harmony of a parse tree 41
4.2 Rules for determining the harmony of a derivation graph 44
4.3 The system of equations that determines W_{root} 52

List of Figures

2.1 An illustration of a simple connectionist network. 18

3.1 A derivation graph of the string *aabb* 30
3.2 A spanning tree of a derivation graph 35

4.1 A parse tree and harmony values for the word *aababa* 42
4.2 A parse tree and harmony values for the word *aabb* 43
4.3 A graphical representation of an energy function 48

5.1 Energy of a harmony network with one positive eigenvalue 84
5.2 Energy of a harmony network with one zero eigenvalue 85
5.3 Energy of a harmony network with two negative eigenvalues 87

Chapter 1

Overture

1.1 The Challenge

In their controversial and influential paper, Fodor and Pylyshyn (1988) examined connectionist or neural network models, and their ability to explain human thought. They note that human thought is systematic. That is, if someone is capable of thinking a thought then they must also be able to entertain a number of other thoughts which are systematically related to the first thought. For example, if a person is able to think that "John loves Mary", then they must *necessarily* also be capable of entertaining the thought "Mary loves John," as well as a host of other similarly related notions. Fodor and Pylyshyn go on to note that humans are able to produce and understand an inexhaustible number of thoughts. Any explanation of cognition must, declare Fodor and Pylyshyn, provide a mechanism whereby this systematicity and productivity cannot fail to arise.

According to Fodor and Pylyshyn, the proper way to achieve this goal is with a Classical model which works by "storing, retrieving, or otherwise operating on structured symbolic expressions." In other words, classical models consist of two parts — a representation part, and an action part. The representation part is a way to internally model concepts in terms of symbols. For example, the sentence "John loves Mary" might have an internal representation loves(john,mary). The action part of a classical model is the way in which the model, or program operates on these

representations — the way the steps in the algorithm manipulate symbol expressions. In the case of the relationship between Mary and John, Mary might decide that John loves her by executing a sequence of actions such as "compare 'mary' with the second argument of the relation 'loves' where 'john' is the first argument." The classical models manipulate structured symbol representations. In other words, the symbols and the structure surrounding them, together with the classical algorithms, cause the classical model to behave in the way it does.

According to Fodor and Pylyshyn, connectionist models do none of these operations on symbolic expressions, and so they are doomed to failure. The failure stems from their assertion that the presence of symbols and a structure imposed on those symbols is inherently necessary to guarantee that a particular model will act appropriately in various circumstances. Fodor and Pylyshyn note that connectionist models do not have these structured symbol representations. In particular, in connectionist systems, objects may have different neural representations depending on their context. Because they lack such structured symbolic representations, connectionist networks cannot perform operations that are sensitive to the structure of a representation, but only to the total representation. Indeed, if the total representation is all that can be discerned, and the symbols themselves are missing, Fodor and Pylyshyn assert that connectionist models cannot consistently perform operations that depend on the constituents of the structure. Consequently, connectionist models will never account for human ability to think systematically, logically and uniformly. Similarly they will not account for the ability to generate and understand an infinite number of thoughts.

Furthermore, say Fodor and Pylyshyn, even if connectionism could model human cognition, the best it will ever do is directly implement a classical or symbolic algorithm. That is, the connectionist system would necessarily represent the symbolic structure, and mimic the classical model's manipulation of the symbols to determine its output. For this reason, Fodor and Pylyshyn say connectionists may as well accept the fact that at best they are only implementing classical algorithms.

1.2 The Answer

In answer to Fodor and Pylyshyn's charges, Smolensky, Legendre and Miyata (Smolensky, Legendre, and Miyata 1992; Smolensky 1993) proposed a mechanism, "Harmony Theory," by which connectionist models can accept formal languages. Thus, the proponents claim that connectionist networks can perform structure sensitive operations without implementing classical algorithms.

Like a classical model, a connectionist network in Harmony Theory possesses a representation of symbolic structure. Each symbol's representation is spread amongst many nodes, and is in fact entwined with the representations of other symbols. Thus, unlike the classical model, the connectionist network does not manipulate the constituents or symbols of the structure. Rather, processing in a connectionist system is a "kind of parallel holistic manipulation of symbolic structures" (Smolensky, Legendre, and Miyata 1992, p9). In other words, classical models directly manipulate individual symbols while harmony theory describes a means by which the symbolic structure is manipulated as a whole. In harmony theory, the individual symbols and their positions in the structure do not *directly* cause the connectionist network to behave the way it does.

In a Harmony Theory connectionist network, or Harmony network, the symbolic structure is represented by a pattern of activation over the nodes in the network. Harmony theory describes a method of determining the interconnections in the harmony network so that the activation value changes into a completion of the computation (Legendre, Miyata, and Smolensky 1990). That is, the distributed pattern changes by virtue of the individual activation values, not by virtue of the symbols themselves. Thus, the algorithm in a Harmony network performs structure-sensitive operations without manipulating the symbols themselves.

More precisely, the representation part corresponds to classical structure-sensitive representations. A variety of mechanisms have been proposed whereby connectionist networks could associate symbols with their syntactic positions. For example, Shastri and Ajjanagadde (Shastri and Ajjanagadde 1993; Tesar and Smolensky 1994) propose to associate objects in the representation by firing neurons synchronously.

Plate (1993) proposes that symbols could be associated via a "holographic" convolution scheme. Harmony theory proposes a similar scheme using tensors to convolve representations for symbols with their syntactic positions. Smolensky, Legendre, and Miyata (1992) demonstrate that these tensor representations can be used to represent parse trees for context free languages.

Smolensky et al. also assert that a similar line of reasoning would allow the representation of derivation graphs of arbitrary type 0 languages (Smolensky, Legendre, and Miyata 1992, p44). The present thesis demonstrates a specific means by which tensor representations could capture such derivation graphs, and discusses the pitfalls of such a representation. We will also develop a second type of tensor representation for derivation graphs which is unrelated to that implied by Smolensky et al.

The action part of harmony theory purportedly describes how a connectionist network could be constructed so that it accepts structured representations of strings in a formal language. Essentially, the idea is to construct a "harmony function," which maps the set of all possible parse trees to the real numbers. The function is defined so that the value of the harmony function for a correct parse tree will be greater than the harmony value for any incorrect parse tree. Smolensky et al. give a method to construct a harmony function for any context free grammar, and suggest that a similar harmony function exists for any type 0 language. The proposed harmony function does not fulfill all the requirements for the construction of a harmony network, and so, we will modify it slightly. The thesis then goes on to show the precise method for constructing a harmony function for a type 0 language.

Smolensky et al. note that some connectionist systems are known to minimize particular functions, called Liapunov or energy functions (e.g. Golden 1986; Hopfield 1982). If a connectionist network can be synthesized so that its energy function is the negation of the harmony function for the parse trees in a particular language then that connectionist network will seek well-formed parse trees for words in that language. Smolensky et al. purportedly define just such a harmony function and connectionist network. Given a grammar on which a harmony function has been defined, they explain how to construct a network so that the network's zero-energy states correspond to parse trees with maximum harmony values. They use this construction to claim

that the network's stable equilibria represent valid parse trees. This thesis shows that their construction is incorrect in that the parse tree representations do not correspond to the network's zero-energy states. Fortunately, as demonstrated in this thesis, the construction is easily corrected.

1.3 The Rebuttal: The Main Result

Fodor and Pylyshyn's classical models must surely work on some machine model — a model that receives input, processes it and produces output. Language theorists recognize that such a machine is equivalent to one that accepts the language consisting of input-output pairs. Consequently, if the classical models can be formalized at all, then they are equivalent to a machine that accepts a formal language. It is just such a machine that Harmony theory claims to construct in answer to Fodor and Pylyshyn's challenge:

> Furthermore . . . we saw that this embedding invariance generates Harmony functions which can be used to express any Context Free Language; that is, the network can distinguish ill- and well-formed strings from such a language. Indeed we saw how this extends to arbitrary formal languages." (Smolensky, Legendre, and Miyata 1992, p44)

The present thesis proves as its main result that this statement is false. More precisely, it shows no network of the type described by Smolensky, Legendre, and Miyata (1992) can distinguish ill- and well-formed strings from any language.

Smolensky et al. do not define what it means for a connectionist system to "distinguish ill- and well-formed strings." This thesis will therefore suggest a reasonable definition for a connectionist decision network. Formalization of the definition will illuminate the fact that inputs to a harmony network are assumed to be parse trees, not strings of symbols to be tested for membership in the language. Furthermore, there is no output to indicate if a string is in the language or not; the network's acceptance or rejection must be inferred from a value which the network itself cannot calculate. Thus, the proposed harmony networks cannot be said to decide an arbitrary language:

the claim that harmony networks can distinguish ill- and well-formed strings from a context free language is false. Later, the present thesis will expand on why such a network is insufficient to meet Fodor and Pylyshyn's challenge.

However, even given that the problems of associating the string with its parse tree and deducing the output can be solved, harmony networks do not work as designed. This thesis shows that harmony networks admit stable equilibria that are not valid parse tree representations. Consequently, harmony theory and tensor representations are unable to support their proponents' claim that connectionist networks can exhibit structure-sensitive processing without directly implementing classical algorithms.

Chapter 2

Background and Notation

2.1 Sets

The notation to be used in this thesis for sets is summarized in Table 2.1 In addition, this thesis uses the following terms when referring to functions:

- **Injection** A function, F, is an *injection* if each element in the domain maps to a unique element of the codomain. That is, if $d_1 \neq d_2$ then $F(d_1) \neq F(d_2)$.
- **Surjection** A function, F, is a *surjection* if for every element c of the codomain, there is an element d in the domain, such that $c = F(d)$
- **Bijection** A function which is both an injection and a surjection is called a *bijection* or an *isomorphism.* If there is an isomorphism between two sets A and B, then we say that A is *isomorphic* to B and write $A \cong B$.
- **Odd** For the purpose of this thesis, an *odd function* is a function, $f : \mathbb{R} \to \mathbb{R}$ that is symmetric about the origin: $f(-x) = -f(x)$.
- **Even** Again, for the purpose of this thesis, an *even function* is a function, $f : \mathbf{R} \to \mathbf{R}$ that is symmetric: $f(-x) = f(x)$.

Notation	Description
$\{a, b\}$	The set consisting of a and b.
$e \in A$	e is an element of a set A.
$e \notin A$	e is not an element of a set A.
$\mathbb{R}$	The real numbers.
$\|A\|$	The size of the set A.
$A \subseteq B$	A is a subset of B.
$A \subset B$	A is a proper subset of B.
$A \cup B$	The union of A and B.
$A \cap B$	The intersection of A and B.
$A \setminus B$	The difference between A and B; those elements that are in A, but not in B.
$\overline{A}$	The complement of A.
$A \times B$	The cartesian product of A and B ; the set of all ordered pairs where the first element is from A and the second element is from B.
A^n	The cartesian product with A by itself n times; the set of n-tuples of elements of A.
$F : D \to C$	F is a function which maps the domain, D to the codomain, C.
$F(c) = d$	The function, F, maps the element c to d.

Table 2.1: Notation for Sets

2.2 Graph Theory

2.2.1 Definition

A *directed graph* is an ordered pair $G = (V, E)$ where V is a set of vertices and $E \subseteq V \times V$ is a set of edges. An edge (u, v) is said to be *incident from* u and *incident to* v, or simply *from* u and *to* v.

An *undirected graph* or *graph* is similar to a directed graph, except that the edge set E contains two-element subsets of V. Thus, in an undirected graph, edges have no "from end" or "to end." We will denote edges in an undirected graph by an ordered pair, (u, v); notice that $(u, v) = (v, u)$.

This thesis will make use of the following terms relating to graphs and directed graphs:

- **Subgraph** A graph $G' = (V', E')$ is a *subgraph* of a graph $G = (V, E)$ if G' is a graph and, $V' \subseteq V$ and $E' \subseteq E$.

- **Path** In a graph (V, E), a sequence of distinct vertices $(v_0, v_1, \ldots, v_k)$ is a *path* from u to v if $u = v_0$, $v = v_k$ and for $i = 1 \ldots k, (v_i - 1, v_i) \in E$.

- **Connected** If there is a path between every pair of vertices in a graph, then the graph is said to be *connected.*

- **Cycle** A *cycle* is a path, $(v_0, v_1, \ldots, v_k)$, which begins and ends at the same place, so $v_k = v_0$. If a graph does not contain any cycles, then it is called *acyclic.*

- **In-degree** In directed graphs, the *in-degree* of a vertex, v is denoted by $deg_{in}(v)$, and indicates the number of edges that are incident to the vertex.

- **Out-degree** In directed graphs, the *out-degree* of a vertex, v is denoted by $deg_{out}(v)$, and indicates the number of edges that are incident from the vertex.

- **Source** If a vertex, s in a directed graph has $deg_{in}(s) = 0$ then it is called a *source* vertex.

- **Sink** If a vertex, s in a directed graph has $deg_{out}(s) = 0$ then it is called a *sink* vertex.

- **Tree** A *tree* is an undirected connected acyclic graph. All the trees considered in this thesis are rooted, ordered trees. A *rooted* tree is a tree where a particular vertex is identified as the *root.* If (u, v) is the last edge in the path from the root of the tree to a vertex v, then u is the *parent* of v and v is u's *child.* If the children of every node in a tree are given an order, then the rooted tree is *ordered.* A *leaf* is a vertex that has no children.

2.3 Formal Languages

For a more comprehensive review of formal languages, the reader is referred to Hopcroft and Ullman's standard text, (Hopcroft and Ullman 1979), which serves as the basis for this section.

2.3.1 Definition

An *alphabet* is a finite set of *symbols*. A *string* or *word* is a sequence of zero or more symbols from a given alphabet. This thesis will use the words "string" and "word" interchangeably. The special word, ϵ, contains no symbols, and is called the *empty word*.

The *length* of a word is simply the number of symbols in the word. If w is a word, then $|w|$ is its length. Thus $|\epsilon| = 0$, and $|abcabc| = 6$.

A *language* is a set of words over a given alphabet. The following are some important languages:

- The empty set, $\emptyset$ is the language that contains no words.
- If Σ is an alphabet, then Σ^i is the set of words of length i made up of any combination of symbols from Σ
- The finite language of all strings over a given alphabet with length at most n is denoted $\Sigma^{\leq n}$.

$$\Sigma^{\leq n} = \bigcup_{i=0}^{n} \Sigma^i$$

- The infinite language of all strings over a given alphabet, Σ is denoted Σ^*.

$$\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$$

2.3.2 Grammars

A *grammar* is a four-tuple, (V, Σ, P, S) that describes how to derive strings in a language. Σ is the alphabet over which the language is defined; when referring to a

grammar, its members are sometimes called *terminals*. V is a set of *non-terminals* such that $V \cap \Sigma = \emptyset$, and S is a special non-terminal, called the *start symbol*. The set of *productions*, $P \subseteq (V \cup \Sigma)^* \times (V \cup \Sigma)^*$, describes how to rewrite strings of non-terminals and terminals.

Each production is written as $\alpha \rightarrow \beta$ which means "replace an occurrence of the string α with β," where α and β are strings of terminals or non-terminals ($\alpha, \beta \in (V \cup \Sigma)^*$). If $\gamma, \delta \in (V \cup \Sigma)^*$ are strings of terminals and non-terminals, and there is a production, $(\alpha \rightarrow \beta) \in P$ then we say $\gamma\alpha\delta$ *directly derives* $\gamma\beta\delta$ or $\gamma\beta\delta$ follows from the *application* of the production to $\gamma\alpha\delta$. We denote this relation by writing $\gamma\alpha\delta \underset{G}{\Rightarrow} \gamma\beta\delta$. If $\alpha_m \in (V \cup \Sigma)^*$ follows from the application of 0 or more productions to the string $\alpha_1 \in (V \cup \Sigma)^*$ then we say α_1 *derives* α_m and write, $\alpha_1 \overset{*}{\underset{G}{\Rightarrow}} \alpha_m$. If the grammar is clear from the context, then the symbols, $\underset{G}{\Rightarrow}$ and $\overset{*}{\underset{G}{\Rightarrow}}$ are written $\Rightarrow$ and $\overset{*}{\Rightarrow}$ respectively.

The set of all strings of terminals that can be derived from the start symbol is called the language *generated* by the grammar. This language is denoted, $L(G)$. Thus if $G = (V, \Sigma, P, S)$ is a grammar, then $L(G) = \{w | w \in \Sigma^* \text{ and } S \overset{*}{\underset{G}{\Rightarrow}} w\}$.

2.3.3 Derivation Graphs

A *derivation sequence* for a word, w is a sequence of strings of terminals and non-terminals such that the first string is the start symbol, the last string is the word, and each intermediate string follows from the previous by the application of one production. More formally, a sequence $w_1, w_2, \ldots, w_n$ is a derivation sequence for w if $w_1 = S$, $w_n = w$ and $w_i \Rightarrow w_{i+1}$, for $i = 1, \ldots, n-1$.

A *derivation graph* is a graphical representation of a derivation sequence. To construct a derivation graph for a word in a language that is generated by a grammar, begin by writing down a derivation sequence for the word, and call the symbols in this derivation sequence the nodes of the graph. Next add arcs joining any symbols that remained unchanged from one step to the next. For each step, add arcs from the symbols of the left hand side of the production rule that was applied in that step to those of the right hand side. Finally, contract edges which begin and end at identical

symbols.

More precisely, let w be a word in the language generated by a grammar, $G = (N, \Sigma, P, S)$. Construct the derivation graph for w by performing the following steps:

1. Write a derivation sequence, $w_1, \ldots, w_n$, for w. Let $l_i = |w_i|$ be the length of the derived string after the ith step. Let $a_{i,j}$ be the jth symbol in w_i, for $i = 1 \ldots n, j = 1 \ldots l_i$. Let p_i, q_i and r_i be such that $a_{i,p_i} \ldots a_{i,q_i}$ are the symbols on the left side of the production applied in the ith step, and $a_{i+1,p_i} \ldots a_{i+1,r_i}$ are the symbols on the right side of that production.

2. Construct a directed graph $H = (V, E)$ where $V = \{a_{i,j} | i = 1 \ldots n, j = 1 \ldots l_i\}$.

3. Add the edges, $(a_{i,j}, a_{i+1,k})$ if the two vertices satisfy one of the following rules:

   - If $p_i \leq j \leq q_i$ and $p_i \leq k \leq r_i$.
   - If $j = k < p_i$.
   - If $q_i < j$, $r_i < k$ and $l_i - j = l_{i+1} - k$.

4. For every edge $(a_{i,j}, a_{i+1,k}) \in E$ if $j < p_i$ or $q_i < j$ then contract $(a_{i,j}, a_{i+1,k})$.

In every case, the resulting graph will be a directed acyclic graph, where the source is the start symbol, S, and the sinks constitute w. For context free and regular languages, derivation graphs are trees and are called *parse trees.*

2.3.4 Machines and Languages

A *machine* is a finite mathematical model of a system that produces a single output upon presentation of a finite input. In this thesis, we are only interested in machines that halt on every input, answering either "yes" or "no." If S is the set of strings to which a machine, M, answers "yes," then M is said to *accept* S. This is denoted, $S = L(M)$, and $L(M)$ is called the *language of* M.

Regular Languages

If the productions in a grammar all have one non-terminal on the left hand side and at most one terminal and one non-terminal (in order) on the right hand side, then the grammar is called a *regular grammar.* The language that is generated by a regular grammar is called a *regular language.*

In other words, let $G = (V, \Sigma, P, S)$ be a grammar where if $\alpha \rightarrow \beta$ is a production in P, then $\alpha \in V$ and $\beta \in (\{xy | x \in \Sigma \; and \; y \in V\} \cup \Sigma \cup V)$. Then G is a regular grammar and $L(G)$ is a regular language.

Context Free Languages

If the productions in a grammar all have one non-terminal on the left hand side and at most two terminals and non-terminals on the right hand side, then the grammar is called a *context free grammar (CFG).* If a language is generated by a context free grammar then it is called a *context free language (CFL).*

Formally, let $G = (V, \Sigma, P, S)$ be a grammar. If for every $(\alpha \rightarrow \beta) \in P$, $\alpha \in V$ and $\beta \in \{\Sigma \cup V\}^{\leq 2}$, then $L(G)$ is a context free language.

Context Sensitive Languages

If every production in a grammar has at least as many terminals and non-terminals on the right hand side as on the left, then that grammar is called a *context sensitive grammar.* Languages that are generated by context sensitive grammars are called *context sensitive languages.*

More formally, let $G = (V, \Sigma, P, S)$ be a grammar where if $(\alpha \rightarrow \beta) \in P$ then $|\alpha| \leq |\beta|$. The language, $L(G)$ is a context sensitive language.

Recursive Languages

If G is an arbitrary grammar, then $L(G)$ is a *recursively enumerable language*, and G is called a *type 0* grammar.

If $L(G)$ is a recursively enumerable language, then there is a machine called a *Turing machine* which answers "yes" whenever the string presented to its input is in

$L(G)$. However, this machine may not halt on every input. If the Turing machine halts on every input, then $L(G)$ is called a *recursive language.*

2.4 Linear Algebra

The notation used in this thesis for linear algebra is summarized in Table 2.2. We will also use the following terms:

- **Linear function** If V is a vector space over a field, X, and $f : V \rightarrow V$ is such that for every $x \in X$ and $a, b \in V$, $f(a + xb) = f(a) + xf(b)$, then f is a *linear function* of V.

- **Linearly independent** If V is a vector space over X, and $A = \{a_1, a_2, \ldots, a_n\} \subset V$ is such that for every $x_1, x_2, \ldots, x_n \in X$, $x_1a_1 + x_2a_2 + \cdots + x_na_n = 0$ implies that $x_1 = x_2 = \cdots = x_n = 0$, then A is a *linearly independent* set of vectors.

- **Direct Sum** If U and V are subspaces of a vector space, W with the property that for every $w \in W$ there exist unique vectors $u \in U$ and $v \in V$ such that $w = u+v$ then W is the *direct sum* of U and V, and we denote this, $W = U \bigoplus V$ (Fisher 1970, p104).

- **Eigenvalue** If $A \in M[n, n]$, $e \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}$ are such that $Ae = \lambda e$ then λ is an *eigenvalue* of A, and e is an *eigenvector* of A.

2.5 Tensors

The tensor product is an operation, satisfying certain properties (see e.g. Yokonuma 1977), and which maps a pair of vector spaces to a third vector space. The tensor product of two vector spaces U, V is written $U \otimes V$, and for $u \in U$ and $v \in V$, $u \otimes v$ denotes the tensor product of u and v.

Notation	Description
V	Usually, we will use capital letters to represent a vector space.
$M[m,n]$	The vector space of $m \times n$ real matrices.
$\vec{a}$	Except in the case of matrices, we will usually use small letters to denote a vector. If there is confusion, we will use the special symbol, $\vec{}$.
$\det \lvert A \rvert$	The determinant of $A \in M[n,n]$.
a^T	The transpose of a

Table 2.2: Notation for Linear Algebra

In this thesis, we are concerned entirely with a very simple tensor product called the Kronecker product (Yokonuma 1977, p17) which maps pairs of matrices to a third matrix. If $U = M[m,n]$ is the vector space consisting of $m \times n$ matrices over the real numbers, and $V = M[m',n']$, then the Kronecker product, $U \otimes V = M[mm',nn']$ is given by

$$A \otimes B = \begin{pmatrix} a_{1,1}B & a_{1,2}B & \cdots & a_{1,n}B \\ a_{2,1}B & a_{2,2}B & \cdots & a_{2,n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1}B & a_{m,2}B & \cdots & a_{m,n}B \end{pmatrix}$$

where $A \in U$ and $B \in V$.

This paper will make use of the following general properties of tensor products:

T1 If U and V are vector spaces over a field F, then $U \otimes V$ is also a vector space over F. This statement is a recapitulation of part of the definition, but it is worth stating again.

T2 Associativity: $(U \otimes V) \otimes W \cong U \otimes (V \otimes W)$ (Yokonuma 1977, p12). If the tensor product is the Kronecker product, and U, V and W are matrices over $\mathbb{R}$, then $(U \otimes V) \otimes W = U \otimes (V \otimes W)$.

T3 Commutativity: $U \otimes V \cong V \otimes U$ (Yokonuma 1977, p10). It may not be the case that $u \otimes v = v \otimes u$. For example, if $U = V = M[1,2]$ are vector spaces over the

reals, and the tensor product is the Kronecker product.

$$\begin{aligned}(1,0)\otimes(0,1) &= (0,1,0,0)\\ (0,1)\otimes(1,0) &= (0,0,1,0)\end{aligned}$$

T4 If U and V are vector spaces, and $v \in V$, then there is a linear mapping $\varphi_v : U \otimes V \rightarrow U$ such that for any $u \in U$, $\varphi_v(u \otimes v) = u$ (Yokonuma 1977, p10).

T5 Let U and V be vector spaces over a field, F, and let $w \in U \otimes V$ be given by

$$w = \sum_{i=1}^{r}(u_i \otimes v_i)$$

where $u_i \in U$ and $v_i \in V$ for $i = 1 \ldots r$. If $\{v_1, v_2, \ldots, v_r\}$ is linearly independent, then the elements $u_1, u_2, \ldots, u_r$ are uniquely determined (Yokonuma 1977, p9).

T6 If $a, b \in M[m,n]$, $c \in M[m',n']$, and the tensor product is the Kronecker product, then

$$(a+b)\otimes c = a\otimes c + b\otimes c$$

Because T2 guarantees the associativity of $\otimes$, it is convenient to use the following shorthand for large tensor products. If V is a vector space, then define:

$$\begin{aligned}T^0(V) &= \{\}\\ T^1(V) &= V\\ T^p(V) &= V\otimes T^{p-1}(V), \text{ for } p>1\\ T(V) &= \bigoplus_{i=0}^{\infty} T^i(V)\end{aligned}$$

The elements of $T^p(V)$ are tensors of *degree* p, and the elements of $T(V)$ are vectors whose pth component is a tensor of degree p. Smolensky, Legendre, and Miyata (1992) assert that the elements of $T(V)$ can be thought of as the concatenation of tensors of increasing degrees.

2.6 Neural Networks

Connectionist or neural network models of computation consist of a large set of simple processing units connected in a network (see for example Rumelhart, Hinton, and McClelland 1986). Each processing unit carries a certain weight or activation level. The external world can influence and perhaps be influenced by the activation level in some or possibly all of the processing units. That is, the input is provided to the network as activation levels on a fixed subset of the units; this subset is called the *input set.* Similarly, the output is read from the network by examining the activation levels of the *output set.* Units that are in neither the input set nor the output set are known as *hidden units.*

Upon receiving an initial activation on its input units, the connectionist network transmits that activation between processing units via the interconnections of the network. Using a weight factor, each connection in the network can modify the activation level that it receives from the sending unit and sends to the receiving unit. The rule that determines exactly how activation in a unit will affect that unit's neighbours is called the *activation rule.* This rule, together with the way the processing units are connected in the network, the weights on the connections, and the units' initial state determines how the neural network will behave.

More formally, let (U, E) be a weighted directed graph, where U is a set of nodes, or processing units, and E is a set of edges, each of which is assigned a weight. We will define a matrix, $W \in M[|U|, |U|]$ called the *weight matrix*; it describes how the nodes in the network are connected. If $W_{i,j}$ is zero, then there is no connection from node U_j to node U_i, and the activation level of U_j will have no direct affect on the activation level of U_i. On the other hand, if $W_{i,j}$ is non-zero, then there is a connection from node U_j to node U_i. The value of $W_{i,j}$ indicates a scale factor that is applied to the activation level that propagates along that link.

For example, suppose a particular network contains three nodes, U_1, U_2, and U_3,

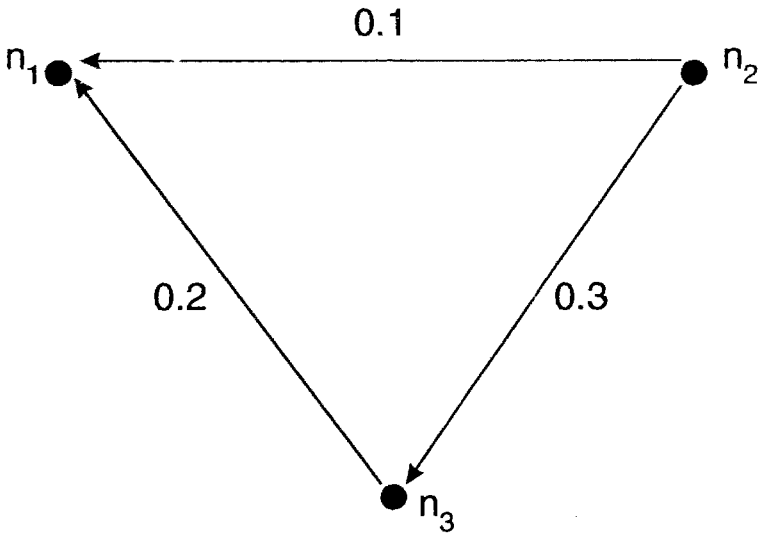

Figure 2.1: An illustration of a simple connectionist network.

and the weight matrix of the network is given by:

$$W = \begin{pmatrix} 0 & 0.1 & 0.2 \\ 0 & 0 & 0 \\ 0 & 0.3 & 0 \end{pmatrix}$$

Then activation in U_2 is transmitted to U_1 and U_3, but is scaled by factors of 0.1 and 0.3 respectively. Similarly, activation in U_3 is transmitted to U_1, but is scaled by a factor of 0.2 by the link between those two nodes. This network is illustrated in figure 2.1.

The activation vector of a network consisting of nodes $U_1, U_2, ..., U_n$ is a vector $a = (a_1, a_2, ..., a_n)$, where $a_i \in \mathbf{R}$ is the activation level of node U_i. In general, the activation levels can vary arbitrarily, but in most models, the range of the activation levels is a finite subset of the real numbers. Because the activation vector changes over time, let $a(t)$ be the activation vector at time t, and let $a_i(t)$ be the ith component of the activation vector at time t. In the above example, the signal received by U_1 at time t is then

$$0.1a_2(t) + 0.2a_3(t)$$

Using the activation rule, U_1 would then have to use this signal to determine how to modify its own activation value;

The activation rule determines how the activation vector changes over time. If the particular connectionist model under consideration is a continuous time model where the processing units update themselves immediately when their neighbours change, as in Hopfield (1984), then the activation rule is a differential equation:

$$\frac{da(t)}{dt} = F(a(t), W)$$

For example, Hopfield (1984) proposes the following activation rule for a continuous time neural model.

$$C_i \frac{du_i}{dt} = \sum_j W_{i,j} a_j - \frac{u_i}{R_i} + I_i \tag{2.1}$$

$$u_i = g_i^{-1}(a_i) \tag{2.2}$$

In this model, C_i, R_i and I_i are biologically motivated constants, and $g_i(x)$ are odd real-valued functions for all i.

On the other hand, if the model is a discrete time model where the entire network is updated during each time step, then the activation rule is iterative:

$$a(t+1) = F(a(t), W)$$

For example, the Brain-state in a box model developed in (Anderson, Silverstein, Ritz, and Jones 1977) represents one of the simplest discrete time models. It uses the following activation rule to determine the next state of the network:

$$b(t) = a(t) + \gamma W a(t) \tag{2.3}$$

$$a(t+1) = S(b(t)) \tag{2.4}$$

Where S() is a vector valued function that truncates each component so that the activation vector always stays within a hypercube centred at the origin; γ is a constant.

Chapter 3

Tensor Representations

As mentioned in Chapter 1, Smolensky, Legendre and Miyata's answer to Fodor and Pylyshyn's challenge consists of two parts: a representation part and an action part. This chapter describes the representation part in detail. The representation part of Harmony theory depends on a technique to relate symbols with their syntactic positions by convolving vectors that represent these symbols and positions via a tensor product. Thus, the representation part of Harmony theory is called *tensor representation.* Starting with the development of simple tensor representations this chapter traces Smolensky, Legendre and Miyata's (Smolensky 1993; Smolensky, Legendre, and Miyata 1992; Smolensky 1990) development of recursive tensor representations, and their use in representing parse trees for context free languages. The chapter then proceeds to develop the use of tensor representations to capture the derivation graphs of Type 0 languages.

3.1 Simple Tensor Representations

If tensor representations are to provide a means to represent structured symbol expressions, they must represent symbols and their syntactic positions. Smolensky, Legendre and Miyata call these syntactic positions *roles* (Smolensky 1990; Smolensky, Legendre, and Miyata 1992). Some examples of roles are the first, second, third and fourth positions in a list, and the relative positions of left-child and right-child of

a vertex in a binary tree. For an example in natural language, consider the sentence "John kissed Mary"; there are three symbols "John" "kissed", and "Mary". Each symbol appears in a particular role. "John", the performer of the action, appears in the agent role, "kissed" appears in the verb role, and "Mary", the receiver of the action, appears in the patient role.

One possible approach to the problem of representing the various symbols in their separate roles is to arbitrarily assign a different representation for each *constituent* or symbol-role pair. So, for example, there would be a representation for each of (John,agent), (kissed,verb), and (Mary,patient). However, this approach is unsatisfactory. As argued effectively by Fodor and Pylyshyn, any representational scheme that can represent sentences such as "John kissed Mary" must also be able to represent "Mary kissed John." If there are different, unrelated representations for (John,agent) and (John,patient), then there is no guarantee that the system that correctly represents John giving a kiss will also be able to represent John receiving a kiss.

Furthermore, if there are different unrelated representations of (John, agent) and (John,patient), then there is no way to relate the "John" in the agent role with the "John" in the patient role. Indeed, if connectionism were to model human intelligence, then the "John" in "John kissed Mary" should also be able to relate to the "John" in "Rachel's husband is John". Otherwise Rachel wouldn't know if she should be upset or not.

Finally, if several units take part in each constituent, so that the representations are fully distributed, then there may be no easy way to untangle the superposition of several constituents. That is, if the entire structure consisting of several constituents is represented by the sum of their individual representations, then some other unrelated constituents could be subsumed by that sum; the result would be that the network would "accidentally" represent two things, one of which would not be true. For example, suppose

$$\begin{array}{ll} \text{(John,agent)} = (1,0,0) & \text{(Fido,agent)} = (0.2,0.4,0.5) \\ \text{(kissed,verb)} = (0,1,0) & \text{(chased,verb)} = (0.5,0.4,0.3) \\ \text{(Mary,patient)} = (0,0,1) & \text{(Felix,patient)} = (0.3,0.2,0.2) \end{array}$$

Then $(1,1,1)$ is the representation for "John kissed Mary" and it is also the representation for "Fido chased Felix". The representation has no way to guarantee that this overlap does not happen because the constituents are assigned arbitrary vectors.

Tensor representations address these issues by assigning some number of real vectors to represent the roles, and other vectors to represent the constituents. The representation of a constituent in a particular role is then the tensor product of the two. For example, suppose the symbol "John" is represented by a vector, $j = (j_1, j_2, j_3)$ and the agent role is represented by $a = (a_1, a_2)$, then the representation of John in the agent role is

$$j \otimes a = (j_1a_1, j_1a_2, j_2a_1, j_2a_2, j_3a_1, j_3a_2)$$

Similarly, if the patient role is represented by $p = (p_1, p_2)$, then the representation of John in the patient role is

$$j \otimes p = (j_1p_1, j_1p_2, j_2p_1, j_2p_2, j_3p_1, j_3p_2)$$

Thus, unlike the arbitrary constituent representations above, a tensor product representation scheme that is able to represent a particular constituent in a given role is necessarily able to represent that constituent in any other role. A tensor representation guarantees that the system that correctly represents John giving a kiss will also be able to represent John receiving a kiss.

The second important difference between the tensor representation and the arbitrary constituent-role representations is that the tensor product allows the retrieval of symbols that have been combined with roles. That is, given a role a, property T4 of tensors furnishes a linear mapping φ_a so that $\varphi_a(j \otimes a) = j$. This property allows computations on tensor representations to recognize that the same constituent is used in two different roles. So for example, it is possible to relate the "John" in "John kissed Mary" to the "John" in "Rachel's husband is John".

Like the arbitrary role-constituent pair representations above, tensor representations are superimposed or added to one another to represent several constituents in different roles. For example, if j, k and m are arbitrary but distinct vector representations for "John", "kissed" and "Mary" respectively, and a, v and p are vector

respectively representations for "agent", "verb" and "patient", then the tensor representation for the sentence "John kissed Mary" is $j \otimes a + k \otimes v + m \otimes p$. Provided the reasonable assumption that no two symbols appear in the same role, this representation avoids the trap that catches arbitrary role-symbol pairs; it is able to untangle the role-symbol pairs from the sum, provided the role vectors, a, v and p are independent. In general, if the role vectors $w_1, w_2, \ldots, w_r$ are linearly independent, and $t = \sum(v_i \otimes w_i)$ then property T5 of tensors guarantees that $v_1, v_2, \ldots, v_r$ are uniquely determined from t. The result is that, as long as the vectors that represent the roles, or the *role vectors*, are linearly independent, two different structures have two different activation vectors. Furthermore, there is a way to determine the constituents from the complete representation.

The linear independence which ensures that different structures have different representations also ensures that those representations are explicit. In other words, the information has not been compressed, lost or otherwise hidden. Indeed, considering the two-dimensional case, this explicitness is obvious if the role vectors are the set $\{(1,0),(0,1)\}$. Section 3.2 will show that these role vectors must be used in the recursive case. A symbol (a_1, a_2) in each of those roles would be represented by,

$$\begin{aligned}(a_1, a_2) \otimes (1,0) &= (a_1, 0, a_2, 0)\\ (a_1, a_2) \otimes (0,1) &= (0, a_1, 0, a_2)\end{aligned}$$

These representations are the same as letting the first and third units represent the symbol in the role $(1,0)$ and letting the second and fourth units represent the symbol in the role $(0,1)$. The representation's transparency is ensured by the role vectors' linear independence which ensures that symbols do not collide in the complete representation.

Linear independence of the role vectors is worrisome because the size of any set of linearly independent vectors in a vector space is bounded by the dimension of the vector space. Since activation vectors are in $\mathbf{R}^n$ the number of possible roles for symbols in an n-node neural network is n. Role vectors are then a limited resource, which implies that connectionist models must carefully assign them to gain the maximum benefit. For example, it would not do to assign role vectors for every position in a list

as that would imply that the activation vector could only represent short lists.

Even so, it does not help to be "nearly" linearly independent as then tensor property T5 simply does not hold; the extent to which it fails is an area of current research (Smolensky, Legendre, and Miyata 1992). Smolensky (1991) presents a "self addressing unbinding procedure" similar to T5 that does not require the role vectors to be linearly independent, but merely that the dot product of any two role vectors be small. Using this unbinding procedure, the retrieved value, u_i of a symbol f_i is

$$u_i = f_i + \sum_{j \neq i} \frac{r_i \cdot r_j}{\|r_i\|^2} f_j$$

where $r_j, j = 1 \ldots r$ are the role vectors and f_j is the vector that represents the constituent in the role represented by r_j. Each incorrect role vector contributes an error proportional to its amount of "non-independence" with the correct role vector. So, if there are many role vectors, or their pairwise dot products are large, then the error in this retrieved value can be overwhelming. For example, if there are $n + 1$ unit-length role vectors and the mean dot product of pairs of roles is $\frac{1}{n}$, then the error in the retrieved value has the same magnitude as the original symbol.

In either case, the number of role vectors is extremely limited. On the one hand, if the role vectors must be independent then their number is bounded by the size of the network. On the other hand, if the self-addressing unbinding procedure is used, then the error is overwhelming for large sets of roles.

3.2 Recursive Tensor Representations

Fortunately, there is evidence that not very many role vectors are needed (Smolensky, Legendre, and Miyata 1992). The trick is to choose roles that can be used recursively so that each role is really a composition of atomic roles. For example, rather than having different roles for each element in a list, as in "first", "second", "third", and so on, use the single role "next"; in that case, the third element of the list is the next of the next of the next (of the zero).

In terms of tensors, given a (small) set of roles $R = \{r_1, r_2, \ldots, r_r\}$ recursively

define new roles $R' = \{r'_1, r'_2, \ldots\}$ by

$$R' = \{r \otimes r'_j | r \in R, r'_j \in R'\}$$

That is, R' is the set of all finite length tensor products of elements of R. For example, if $R = \{r_0, r_1\}$ then $R' = \{r_0, r_1, r_0 \otimes r_0, r_0 \otimes r_1, r_1 \otimes r_0, r_1 \otimes r_1, r_0 \otimes r_0 \otimes r_0, \ldots\}$ (Smolensky, Legendre, and Miyata 1992). In the list example above, if the symbol, f is in the list and the "next" role is represented by the vector r, then the representation of f occupying the third position in the list is $f \otimes r \otimes r \otimes r$.

The following theorem, which is a straightforward consequence of tensor property T4, shows that recursive tensor representations can be "unwound" just like the simple, non-recursive representations. That is, given a recursive tensor representation of a constituent, the symbol and role of that constituent can be determined.

Theorem 1 *Let V and R be vector spaces. Let $r = r_1 \otimes r_2 \otimes \ldots \otimes r_k$, where $r_i \in R$, for $i = 1 \ldots k$. Then there is a function, $\varphi_r : V \otimes T^k(R) \rightarrow V$ such that if $v \in V$ then $\varphi_r(v \otimes r) = v$.*

Proof: *(by induction on k) Let $\varphi_{r_i} : (V \otimes T^{i-1}(R)) \otimes R \rightarrow V \otimes T^{i-1}(R)$ be a function that for $a \in V \otimes T^{i-1}(R)$, $\varphi_{r_i}(a \otimes r_i) = a$ (Such a function is guaranteed by property T4 of tensors). Now if $k = 1$, then $\varphi_r = \varphi_{r_1}$ since $\varphi_{r_1}(v \otimes r_1) = v$ for any $v \in V$.*

Assume that for $k = l$, there is a function, $\varphi_{r_{(1,l)}} : V \otimes T^l(R) \rightarrow V$ such that for any $v \in V$, $\varphi_{r_{(1,l)}}(v \otimes r_1 \otimes \ldots \otimes r_l) = v$.

Then, if $k = l + 1$, let $f : (V \otimes T^l(R)) \otimes R \rightarrow V$ be defined by

$$f = \varphi_{r_{(1,l)}} \circ \varphi_{r_{l+1}}$$

Then,

$$\begin{aligned} f(v \otimes r_1 \otimes \ldots \otimes r_l \otimes r_{l+1}) &= \varphi_{r_{(1,l)}}(\varphi_{r_{l+1}}(v \otimes r_1 \otimes \ldots \otimes r_{l+1})) \\ &= \varphi_{r_{(1,l)}}(v \otimes r_1 \otimes \ldots \otimes r_l) \\ &= v \end{aligned}$$

Now, by property T2, the associativity of tensor products,

$$V \otimes T^{l+1}(R) \cong (V \otimes T^{l}(R)) \otimes R$$

Let $i : V \otimes T^{l+1}(R) \rightarrow (V \otimes T^{l}(R)) \otimes R$ *be that isomorphism, then*

$$\varphi_{r_{(1,l+1)}} = f \circ i$$

is a function that maps $V \otimes T^{l+1}R$ *to* V *such that* $\varphi_{r_{(1,l+1)}}(v \otimes r_1 \otimes \ldots \otimes r_{l+1}) = v$

So, by induction, for any k, there is a function $\varphi_r = \varphi_{r_{(1,k)}}$ *such that* $\varphi_r : V \otimes T^k(R) \rightarrow V$ *and for any* $v \in V$, $\varphi_r(v \otimes r) = v$. □

As in the simple tensor representation, this theorem allows computations on tensor representations to recognize that the same symbol is used in two different roles. For example, if an element appeared in the third and fifth position of a list, it would be possible to relate the two instances of the symbol.

To combine two constituents to represent a structure where one symbol fills one role, and another symbol fills another, the tensor products are simply added. Because they are not the same vector space, vector addition is not defined between elements of $V \otimes V$ and $V \otimes V \otimes V$, (or between any $T^p(V)$ and $T^q(V)$, $p \neq q$). For this reason, recursive tensor representations are taken to be elements of $T(V)$ — the infinite direct sum of tensor powers of V.

Recall that the elements of $T(V)$ can be thought of as the concatenation of tensors of increasing degrees (Smolensky, Legendre, and Miyata 1992). If two symbols appear at different levels of recursion, then they can be separated easily by taking the parts of the superposition vector that correspond to the appropriate levels of recursion. That is, if $f_1 \otimes r_1 \in T^p(V)$ and $f_2 \otimes r_2 \in T^q(V)$ are recursive tensor representations of two symbols, f_1, f_2 in recursive roles r_1,r_2, and $p \neq q$ then $t = f_1 \oslash r_1 + f_2 \oslash r_2$ can be decomposed easily into its two summands; to get this decomposition, simply take the components of t that correspond to $T^p(V)$ and $T^q(V)$ and apply the functions implied by Theorem 1 to each. However, it remains to be seen whether the superposition vector, t can be decomposed if both symbols appear at the same level of recursion; that is, if $p = q$.

Theorem 2 shows that such sums of recursive tensor products can be decomposed into their summands. It shows that by applying the function implied by property T5 of tensors iteratively, the symbols in recursively applied roles can be determined.

Theorem 2 *Let A be a linearly independent set of vectors from a vector space, V If $s \in V \otimes T^p(V)$ is given by*

$$s = \sum_{i=1}^{r} u_i \otimes v_{i,1} \otimes v_{i,2} \otimes \ldots \otimes v_{i,p}$$

where $u_i \in V$ and $v_{i,j} \in A$ for $i = 1 \ldots r, j = 1 \ldots p$. Then the elements $u_1, u_2, \ldots, u_r$ are uniquely determined.

Proof: *(by induction on p). The theorem holds for $p = 1$, by property T5 of tensors. Assume that it holds for $p \leq k$. Then for $p = k + 1$,*

$$s = \sum_{i=1}^{r} u_i \otimes v_{i,1} \otimes v_{i,2} \otimes \ldots \otimes v_{i,k+1}$$

and therefore $s \in V \otimes T^{k+1}(V)$. Let $\tau : (V \otimes T^k(V)) \otimes V \rightarrow V \otimes T^{k+1}(V)$ be an isomorphism, whose existence is guaranteed by property T2. Let $w_i = u_i \otimes v_{i,1} \otimes \ldots \otimes v_{i,k}$ be in $V \otimes T^k(V)$ Then

$$s = \tau(\sum_{i=1}^{r} w_i \otimes v_{i,k+1})$$

from which w_i are uniquely determined for $i = 1 \ldots r$, by property T5. By hypothesis, $u_i, i = 1 \ldots r$ are uniquely determined from w_i. So $u_i, i = 1 \ldots r$ are uniquely determined from s. Therefore the claim holds for all p. □

Theorem 2 shows that, like simple tensor representations, recursive tensor representations can be superimposed (added) on one another to represent several symbols in different recursive roles. Yet, even with such superposition, the original symbol-role pairs can be reconstructed. The theorem also shows how this reconstruction can be performed recursively.

Note, however, that if the self addressing unbinding procedure discussed in Section 3.1 and in (Smolensky 1991) is used to unbind the representations then the error due to the role vectors' linear dependence is even more important. If the role vectors are

linearly independent, then the error is zero, otherwise, it depends on the number of role vectors, and the dot product between pairs of them. In a recursive representation, the self addressing unbinding procedure is applied iteratively to its result to regain a symbol. Thus, the error present after the first unbinding is used to determine the second unbinding, and so on; consequently the errors compound as the recursive representation is unwound. The linear independence of the role vectors therefore becomes even more important.

Aside from the linear independence of the vectors, the development of tensor representations (Smolensky 1990; Smolensky, Legendre, and Miyata 1992) has consistently glossed over the fact that tensor products are only defined for vector spaces over fields of characteristic zero such as the real numbers or the rationals. Activation vectors, on the other hand are usually defined over a bounded subset of the reals, such as $[0 \ldots 1]$; certainly, in the models that are required by Harmony theory, the activation values are defined over a bounded subset of the real numbers. In order to represent structures, the tensor representations of constituents with the same dimensionality are added together as vectors. However, this addition may not be possible if every component of the sum must fall in a bounded subset of the reals. In the simple non-recursive case, where there can only be a small number of role-constituent pairs in any representation, this difficulty can be finessed by simply making the components of all the vectors small. However, such is not the case in the recursive tensor representation.

In a recursive representation, because the activation vector could be the sum of a large number of tensor product vectors, the activation values can become saturated. In other words, some units may be required to attain an activation value that is beyond their capacity. Suppose for instance, $u = \sum_i v_{i,1} \otimes v_{i,2} \otimes \ldots \otimes v_{i,k} \otimes v_{i,k+1}$, if there are r role vectors, then there are as many as r^k tensors in the sum. If the smallest component is ϵ, then all the components of each of the tensors have size at least ϵ^{k+1}, and so every component of the sum u is at least $u_i \geq \epsilon^{k+1} r^k = (\epsilon r)^k \epsilon$. Therefore, recursive roles of depth k give representations that simply cannot be represented by the activation vector for all k such that:

$$k \geq \log_{\epsilon r} \frac{1}{\epsilon}$$

The result is that the depth of the recursion is limited not only by the size of the network, but by the chosen representation as well. This means that models, such as harmony networks, that take advantage of recursion to define families of networks that manipulate recursive tensor representations must be careful to avoid saturating the connectionist units. The only way to avoid this saturation without limiting the depth of the recursion is by making the role vectors unit vectors along the axes. If the role vectors are among $\{(1,0,0,\ldots),(0,1,0,\ldots),(0,0,1,\ldots)\}$ then no two summands will use the same unit, and so, no unit will saturate.

3.3 Representation of Parse Trees

Trees are an especially useful example of a recursive structure. Many relationships can be represented using trees. For example, trees can be used to represent relationships between classes and their subclasses, or between objects and their components. In particular, trees can be used to represent the application of production rules from context free grammars (CFGs).

Suppose $G = (V, \Sigma, P, S)$ is a CFG. Then, as described in Section 2.3, for any $w \in L(G)$, we can create at least one tree that represents the application of production rules from G to arrive at w in the leaves.

For example, let $G = (\{S, \Gamma_{1,1}, A, \Gamma_{2,1}, B, \Gamma_{3,1}, C, D\}, \{a, b\}, P, S)$ be a CFG where P is the set of productions:

$$\begin{array}{lll} S \rightarrow \Gamma_{1,1} & \Gamma_{1,1} \rightarrow CB & A \rightarrow \Gamma_{3,1} \\ \Gamma_{3,1} \rightarrow CB & B \rightarrow \Gamma_{2,1} & B \rightarrow b \\ \Gamma_{2,1} \rightarrow AD & C \rightarrow a & D \rightarrow b \end{array}$$

It can be shown that $L(G) = \{a^i b^i | i > 0\}$. Figure 3.1 illustrates a tree that represents the application of the production rules to arrive at $w = aabb$:

Smolensky, Legendre, and Miyata (1992) describe the following tensor representation for binary trees, which is based on a breadth-first search of the tree: Let r_l and r_r be the two role vectors corresponding to the role of left child and right child respectively. If t_l and t_r are the superposition tensors representing the left and right

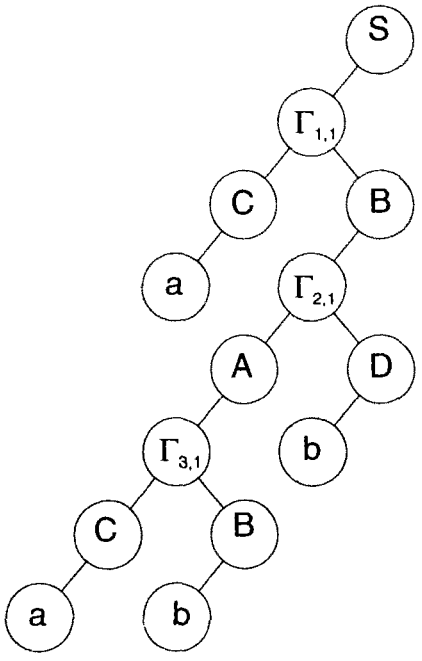

Figure 3.1: A derivation graph of the string *aabb* using rules from the grammar, G of the text

subtrees of a node labeled v, then $s = v + t_l \otimes r_l + t_r \otimes r_r$ is the superposition tensor of the subtree rooted at v. For example, the tree shown in Figure 3.1 would have a tensor representation given by the following construction which follows a breadth-first traversal of the tree:

$$
\begin{aligned}
s &= S + < \text{subtree at } \Gamma_{1,1} > \oslash r_l \\
&= S + (\Gamma_{1,1} + < \text{subtree at } C > \oslash r_l + < \text{subtree at } B > \oslash r_r) \oslash r_l \\
&= S + ((\Gamma_{1,1} + (C + a \otimes r_l) \otimes r_l + (B + < \text{subtree at } \Gamma_{3,1} > \oslash \\
&\quad r_l) \otimes r_r) \otimes r_l \\
&= S + ((\Gamma_{1,1} + (C + a \otimes r_l) \otimes r_l + (B + (B + \\
&\quad < \text{subtree at } A > \oslash r_l + < \text{subtree at } D > \oslash r_r) \otimes r_l) \oslash r_r) \otimes r_l \\
&= S + ((\Gamma_{1,1} + (C + a \otimes r_l) \otimes r_l + (B + (B + (A + \\
&\quad < \text{subtree at } \Gamma_{2,1} > \oslash r_l) \otimes r_l + (D + b \otimes r_l) \oslash r_r) \otimes r_l) \oslash r_r) \otimes r_l \\
&= S + ((\Gamma_{1,1} + (C + a \otimes r_l) \otimes r_l + (B + (B + (A + (\Gamma_{2,1} +
\end{aligned}
$$

$$
\begin{aligned}
& < \text{subtree at } C > \otimes r_l + < \text{subtree at } B > \otimes r_r) \otimes \\
& r_l) \otimes r_l + (D + b \otimes r_l) \otimes r_r) \otimes r_l) \otimes r_r) \otimes r_l \\
= \; & S + ((\Gamma_{1,1} + (C + a \otimes r_l) \otimes r_l + (B + (B + (A + (\Gamma_{2,1} + \\
& (C + a \otimes r_l) \otimes r_l + (B + b \otimes r_l) \otimes r_r) \otimes r_l) \otimes r_l + \\
& (D + b \otimes r_l) \otimes r_r) \otimes r_l) \otimes r_r) \otimes r_l
\end{aligned}
$$

By property T6 of the Kronecker product,

$$
\begin{aligned}
s \; = \; & S + \Gamma_{1,1} \otimes r_l + C \otimes r_l \otimes r_l + a \otimes r_l \otimes r_l \otimes r_l + \\
& B \otimes r_r \otimes r_l + \Gamma_{3,1} \otimes r_l \otimes r_r \otimes r_l + A \otimes r_l \otimes r_l \otimes r_r \otimes r_l + \\
& \Gamma_{2,1} \otimes r_l \otimes r_l \otimes r_l \otimes r_r \otimes r_l + C \otimes r_l \otimes r_l \otimes r_l \otimes r_l \otimes r_r \otimes r_l + \\
& a \otimes r_l \otimes r_l \otimes r_l \otimes r_l \otimes r_l \otimes r_r \otimes r_l + B \otimes r_r \otimes r_l \\
& \otimes r_l \otimes r_l \otimes r_r \otimes r_l + b \otimes r_l \otimes r_r \otimes r_l \otimes r_l \otimes r_l \otimes r_r \otimes r_l + \\
& D \otimes r_r \otimes r_l \otimes r_r \otimes r_l + b \otimes r_l \otimes r_r \otimes r_l \otimes r_r \otimes r_l
\end{aligned}
$$

The previous section showed that provided r_l and r_r are independent, the tensor representation for any tree will be decomposable, and the labels of the vertices will be retrievable, along with their positions in the tree. The result of this tensor representation for trees is that activation vectors can then describe correct (and incorrect) parse trees for words in a context free language. Harmony theory describes how to construct a connectionist network the stable equilibria of which are supposed to be tensor representations of parse trees. This theory is discussed in Chapter 4.

For now, observe that even if activation vectors can represent parse trees for context free languages, it is still "unclear whether this sort of apparatus is adequate to represent all the semantically relevant syntactic relations that Classical theories express" (Fodor and McLaughlin 1990, p344). Indeed, it is certain that the tensor representation leaves a lot of useful languages unexplained. For example, the language $L = \{ww | w \in \{a, b\}^*\}$ is a context-sensitive language, not a context-free language; because productions in a grammar for such a language can have several symbols on the left, a machine that constructs derivation graphs for strings in L will need a more powerful representation than trees.

3.4 Representation of Derivation Graphs

To correctly parse context sensitive languages such as L above, requires directed acyclic graphs. This section begins by discussing a normal form for Type 0 grammars that restricts the number of symbols in a production in the grammar. Such a normal form is required by both the restrictive recursive tensor representation of directed acyclic graphs (DAGs) that is developed later in this section, and by the chapter on harmony theory that follows. The question of whether or not there are connectionist networks that can decide if a decomposition graph is correct for a particular type 0 language is considered in Chapter 4.

3.4.1 Normal Form for Type 0 Grammars

In general, grammars can have productions of any length. However, it is not difficult to modify an unrestricted grammar so that each production has at most three symbols. To construct this normal form for an unrestricted grammar, $\Omega = (V, \Sigma, P, S)$, construct a new grammar, $\Omega' = (V \cup \Gamma, \Sigma, P', S)$. For every production $p_i \in P$:

$$p_i = \alpha_1 \alpha_2 \ldots \alpha_s \rightarrow \beta_1 \beta_2 \ldots \beta_t$$

where $\alpha_j \in V \cup \Sigma, j = 1 \ldots s$, $\beta_k \in V \cup \Sigma, k = 1 \ldots t$ and $s + t \geq 3$, add to Γ new non-terminals $\Gamma_{i,j}$ for $j = 1 \ldots s + t - 1$, and add to P' the following productions:

$$\begin{aligned}
\alpha_1 &\rightarrow \Gamma_{i,1} \\
\Gamma_{i,j}\alpha_{i,j+1} &\rightarrow \Gamma_{i,j+1}, \text{for} j = 1 \ldots s - 1 \\
\Gamma_{i,j} &\rightarrow \beta_{j-s+1}\Gamma_{i,j+1}, \text{ for } j = s \ldots s + t - 2 \\
\Gamma_{i,s+t-1} &\rightarrow \beta_t
\end{aligned}$$

If $s + t < 3$ then add p_i to P. It can be shown that $L(\Omega') = L(\Omega)$ This normal form is important because it establishes a bound on the number of parents and children of each node that the decomposition graph has.

Original Production	New Productions
$S \rightarrow ACaB$	$S \rightarrow \Gamma_{1,1}$ $\Gamma_{1,1} \rightarrow A\Gamma_{1,2}$ $\Gamma_{1,2} \rightarrow C\Gamma_{1,3}$ $\Gamma_{1,3} \rightarrow a\Gamma_{1,4}$ $\Gamma_{1,4} \rightarrow B$
$Ca \rightarrow aaC$	$C \rightarrow \Gamma_{2,1}$ $\Gamma_{2,1}a \rightarrow \Gamma_{2,2}$ $\Gamma_{2,2} \rightarrow a\Gamma_{2,3}$ $\Gamma_{2,3} \rightarrow a\Gamma_{2,4}$ $\Gamma_{2,4} \rightarrow C$
$CB \rightarrow DB$	$C \rightarrow \Gamma_{3,1}$ $\Gamma_{3,1}B \rightarrow \Gamma_{3,2}$ $\Gamma_{3,2} \rightarrow D\Gamma_{3,3}$ $\Gamma_{3,3} \rightarrow B$
$CB \rightarrow E$	$C \rightarrow \Gamma_{4,1}$ $\Gamma_{4,1}B \rightarrow \Gamma_{4,2}$ $\Gamma_{4,2} \rightarrow E$
$aD \rightarrow Da$	$a \rightarrow \Gamma_{5,1}$ $\Gamma_{5,1}D \rightarrow \Gamma_{5,2}$ $\Gamma_{5,2} \rightarrow D\Gamma_{5,3}$ $\Gamma_{5,3} \rightarrow a$
$AD \rightarrow AC$	$A \rightarrow \Gamma_{6,1}$ $\Gamma_{6,1}D \rightarrow \Gamma_{6,2}$ $\Gamma_{6,2} \rightarrow A\Gamma_{6,3}$ $\Gamma_{6,3} \rightarrow C$
$aE \rightarrow Ea$	$a \rightarrow \Gamma_{7,1}$ $\Gamma_{7,1}E \rightarrow \Gamma_{7,2}$ $\Gamma_{7,2} \rightarrow E\Gamma_{7,3}$ $\Gamma_{7,3} \rightarrow a$
$AE \rightarrow \epsilon$	$A \rightarrow \Gamma_{8,1}$ $\Gamma_{8,1}E \rightarrow \Gamma_{8,2}$ $\Gamma_{8,2} \rightarrow \epsilon$

Table 3.1: Normal form for the grammar for $L = \{a^i | i$ is a positive power of $2\}$

For example consider the language, $L = \{a^i | i$ is a positive power of $2\}$. The productions for a grammar, $G = (\{S, A, B, C, D, E\}, \{a\}, P, S)$ for L are reproduced from (Hopcroft and Ullman 1979, p220) below:

$$\begin{array}{lll} S \rightarrow ACaB & Ca \rightarrow aaC & CB \rightarrow DB \\ CB \rightarrow E & aD \rightarrow Da & AD \rightarrow AC \\ aE \rightarrow Ea & AE \rightarrow \epsilon & \end{array}$$

The productions in the normal form are shown in Table 3.1.

3.4.2 Spanning Tree Representation of Directed Acyclic Graphs

The previous section demonstrates that in order to show that tensor representations can capture the derivation graph of a recursive language, it is necessary to show that they can represent directed acyclic graphs where each node has at most two inbound and two outbound edges. The approach suggested by Smolensky, Legendre

and Miyata's development of tree representations is to create a number of parent roles and a number of child roles. The construction will employ these roles to represent a spanning tree of the graph as before, and then to represent the non-tree edges separately.

The Representation

Let $G = (V, E)$ be a directed acyclic graph. Assume that G has a single source, $s \in V$. Assume also that the in-degree and out-degree of each node is bounded, that is for each $v \in V$, $deg_{\text{in}}(v) \leq d_i$ and $deg_{\text{out}}(v) \leq d_o$. The previous section showed that d_i and d_o can be as low as 2 for the derivation graphs of recursive languages. If there is an arbitrary order ($<$) applied to the vertices of G , then there is a natural tensor representation for G.

Let $E' = \{(u, v) | \forall u' \in V, (u', v) \in E \Rightarrow u \leq u'\}$ be the set of "least" edges into each vertex. Now E' covers $V \setminus \{s\}$ since every vertex $v \neq s$ has a least u such that $(u, v) \in E$ for otherwise G has more than one source. Because G is acyclic, E' also covers s. Let $T = (V, E')$ be the subgraph of G that is induced by E'. T is a tree because no vertex has more than one edge coming into it, and because G has a single source and no cycles. For example, in Figure 3.2, the darkened edges represent T.

We can consider the tree to be rooted at s, and ordered by the original ordering on the vertices. Now, let $\Phi = \{r_1, r_2, \ldots, r_m\}$ be role vectors, where r_i corresponds to the role of ith subtree. Then in a manner similar to the representation of binary trees in the previous section, there is a recursive tensor representation for T. It remains to be seen how the edges in $E \setminus E'$ are represented.

Let $\Psi = \{p_1, p_2, \ldots, p_n\}$ be role vectors, where p_i corresponds to the role of ith parent (in the order $(V, <)$). Suppose that

$$\begin{aligned} t_u &= l_u \otimes r_{u,1} \otimes r_{u,2} \otimes \ldots \otimes r_{u,j} \\ &= l_u \otimes r_u \\ t_v &= l_v \otimes r_{v,1} \otimes r_{v,2} \otimes \ldots \otimes r_{v,k} \\ &= l_v \otimes r_v \end{aligned}$$

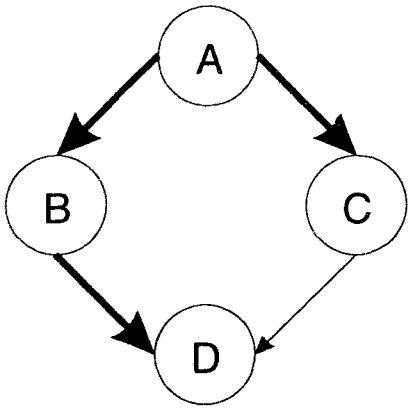

Figure 3.2: A graph with the nodes ordered left to right, top to bottom. The spanning subtree T in the text is represented by the heavy edges.

are the tensor representations for nodes u and v in T, and l_u and l_v are their labels. Now to represent the edge (u, v) in $E \setminus E'$, where u is v's ith parent use:

$$t_{(u,v)} = l_u \otimes r_u \otimes p_i \otimes r_v$$

The representation for the graph, G is then

$$t_G = \sum_{u \in V} (l_u \otimes r_u) + \sum_{\substack{(u,v) \in (E \cap \widetilde{E'}) \\ \text{and} \\ u \text{ is } v' \text{s } i\text{th parent}}} l_u \otimes r_u \otimes p_i \otimes r_v$$

Now, provided that the set of all role vectors, $\Psi \cup \Phi$, is linearly independent, s_G can be decomposed into its constituent parts. Also, due to the uniqueness in tensor property T5, each tensor representation corresponds to a unique (up to graph isomorphism) graph.

An Example

Consider the graph illustrated in Figure 3.2. Its source is A, and its spanning subtree is the set of darkened edges. The representation of the spanning subtree is,

$$\begin{aligned} s_T &= A + < \textit{subtree at } B > \otimes r_l + < \textit{subtree at } C > \otimes r_r \\ &= A + (B + < \textit{subtree at } D > \otimes r_r) \otimes r_l + C \otimes r_r \\ &= A + (B + D \otimes r_r) \otimes r_l + C \otimes r_r \\ &= A + B \otimes r_l + D \otimes r_r \otimes r_l + C \otimes r_r \end{aligned}$$

Now the edge (C, D) is represented by putting the label C in the role constructed by multiplying the roles of C and D with the parent role:

$$\begin{aligned} s_{(C,D)} &= C \otimes r_C \otimes p_l \otimes r_D \\ &= C \otimes r_r \otimes p_l \otimes r_r \otimes r_l \end{aligned}$$

The complete representation for the graph is

$$\begin{aligned} s_G &= s_T + s_{(C,D)} \\ &= A + B \otimes r_l + D \otimes r_r \otimes r_l + C \otimes r_r + C \otimes r_r \otimes p_l \otimes r_r \otimes r_l \end{aligned}$$

Analysis

Despite its uniqueness property, and the fact that the representation can be decomposed into its constituent parts, this method for representing DAGs is unsatisfying. In contrast to the representation of trees, this representation of DAGs could allow the graphs themselves to be malformed. The parent pointer can point to a node that may not actually be there, or it may have a different label in the tree than it does in the non-tree part of the representation. In order to ensure that the DAG is valid, the non-tree edges tacked on the end of the representation will require special treatment in any scheme to develop such derivation graphs, or even in a scheme to recognize a valid derivation graph.

3.4.3 Recursive Representation of Directed Acyclic Graphs

The representation of trees is smooth because trees offer a natural recursive structure: a tree can be constructed from an existing tree by adding a single vertex, and joining that vertex to another vertex already in the tree by the addition of a single edge. The result is that for any vertex in the tree, there is exactly one path from that vertex to the root of the tree. Indeed, there is exactly one path between any two vertices. In general, DAGs cannot be constructed in the same way because for each particular vertex, there may be more than one path from the source to that vertex. When a new

vertex is added to a DAG, several edges may need to be added in order to link several paths. Fortunately, every derivation graph as developed before can be constructed recursively by starting with the source and adding vertices and at most two edges to each new vertex from the existing set of vertices.

The Representation

Proskurowski (1981) proposed a recursive representation for k-trees which serves as an inspiration for a tensor representation of DAGs. The representation is a sequence of sets, where each set represents a vertex and contains the indices of the vertices to which that vertex is joined. In the tensor representation of derivation graphs, we are not only interested in the label of the new vertex but in the position of the added edges (ie whether it is joining a left-parent or a right parent); the tensor representation must include that information as well. As in the representation of the trees, the label and a subgraph will fill each role, and the role will indicate the position of the subgraph.

The proposed representation is a recursive one. If a vertex labeled L is added to the graph which has superposition vector t_G, then the superposition vector of the resulting graph is

$$t'_G = t_G \otimes (l \otimes p_l + r \otimes p_r) + L$$

Here l and r are simple roles indicating "left parent" and "right parent" respectively, and p_l and p_r are recursive roles that indicate the indices of the left and right parents respectively. Note that each time a new vertex is added to the graph, the existing graph's role gains another l and r, so no two subgraphs fall in the same role. Note also that a graph's representation will depend on the order in which the vertices are added.

These indexing roles might be recursively formed from a vector that indicates "next" as in a representation for a list; in that case, they would start at the first node in the graph and count toward the current node (that labelled L). Because we are interested in using this representation to build derivation graphs, it will be more efficient to start with the current node and count backwards. Thus, if p is a simple role that indicates "previous", then if the left parent of the new node was the most

recently added node, and the right parent were the node that was added before that, the role of s_G would be

$$(l \otimes p + r \otimes p \otimes p)$$

An Example

Consider again the directed graph illustrated in Figure 3.2. The representation of this graph is best understood by following the construction of the graph:

$$\begin{aligned}
s_{<A>} &= A \\
s_{<A,B>} &= A \otimes (r \otimes p) + B \\
s_{<A,B,C>} &= (A \otimes (r \otimes p) + B) \otimes (l \otimes p \otimes p) + C \\
s_{<A,B,C,D>} &= ((A \otimes (r \otimes p) + B) \otimes (l \otimes p \otimes p) + C) \otimes \\
&\quad (l \otimes p \otimes p + r \otimes p) + D
\end{aligned}$$

Analysis

The use of indexing roles results in some of the same problems as the tree-based representation. For example, there is no built in limit to the index. Thus, the index could refer to a vertex that is not in the graph. In the representation of trees in Section 3.3, there was no need for an index — it was implicit, the newly added vertex was always joined to the roots of its subtrees. However, in any representation of a graph, there must be some means to refer to vertices inside the graph, for otherwise, there is no way to build multiple paths. Indexing is almost guaranteed in any tensor representation of directed acyclic graphs.

On the other hand, the recursive representation of directed acyclic graphs is uniform. Unlike the tree-based representation, there are no special edges which are appended to the end of the representation. That means that a mechanism that must decide if a graph is valid does not need to have special cases to handle some of the edges.

Chapter 4

Harmony

In Chapter 3, we traced Smolensky, Legendre and Miyata's development of tensor product representations for parse trees for Context Free Grammars. We went on to extend that development to representations of derivation graphs for type 0 grammars. Together, these results show how the activation vector of a connectionist network could represent the steps of an algorithm. Smolensky, Legendre and Miyata's Harmony theory, which is the subject of this chapter, claims to show how a connectionist system could be synthesized so that it naturally "relaxes" into such a state.

Harmony theory consists of two viewpoints or formulations: a symbolic viewpoint and a numeric viewpoint. These viewpoints claim to explain the apparent difference between classical algorithms for intelligence and the computational models postulated by connectionists. The symbolic viewpoint consists of a "harmony function" which maps the set of derivation graphs that are possible over a grammar to the real numbers. A symbolic computation in this formulation is simply a choice of a particular derivation graph that happens to maximize the harmony. The numeric viewpoint consists of an explanation of the evolution of activation vectors in connectionist systems from the point of view of energy minimization. The two are apparently tied together in a procedure to synthesize neural networks that seek maximum harmony computations.

4.1 Symbolic Formulation

Because the tensor representations discussed in Chapter 3 are based on symbols in various roles, a harmonic account of tensor representations starts with a symbolic account. The symbolic harmony of a set of constituents is defined to be the sum of the harmonies of the constituents, taken in pairs:

$$H = \sum_{i \neq j} H_{c_i r_i ; c_j r_j} \tag{4.1}$$

So for example, the harmony of a structure such as "John kissed Mary" would be something like

$$\begin{aligned} H_{((\text{John, agent}),(\text{kissed, verb}),(\text{Mary, patient}))} = \; & H_{((\text{John, agent}),(\text{kissed, verb}))} + \\ & H_{((\text{kissed, verb}),(\text{Mary, patient}))} + \\ & H_{((\text{John, agent}),(\text{Mary, patient}))} \end{aligned}$$

and we would expect this value to be high in the part of Rachel's brain that decides jealousy. If languages such as the set of English sentences that make Rachel jealous are computable at all, then they must be equivalent to some (perhaps restricted) recursively enumerable language (provided Church's thesis holds) (Hopcroft and Ullman 1979, p221). So, we require a harmony function that decides membership of strings in some recursively enumerable languages.

4.1.1 Context-Free Parse Trees

Smolensky (1993) proposed a harmony function for derivation trees for context free grammars in a normal form. That is, their harmony function takes as input a candidate parse tree for a string in a context free language and assigns a harmony value to the tree. If the harmony of the tree is zero, then the string is judged to be in the language. The function examines the labels of the nodes in the derivation tree, and compares parents to their children. The rules that are used to derive this function for any given language are summarized in Table 4.1.

That these rules yield a zero harmony value for well formed trees can be seen by examining the proof presented in (Smolensky 1993). The essential idea of this proof is

Type 0 Production or Symbol	Change in Harmony
$l(v) = a \in \Sigma$	add -1 to H
$l(v) = A \in N$	add -2 to H
$l(root) = S$	add $+1$ to H
$l(v) = \gamma \in \Gamma$	add -3 to H
$(u, v) \in E, v$ the left child of u and $(l(u) \rightarrow l(v)\alpha) \in P$ or $(l(u) \rightarrow l(v)) \in P$, $\alpha \in \Sigma \cup N \cup \Gamma$	add $+2$ to H
$(u, v) \in E, v$ the second child of u and $(l(u) \rightarrow \alpha l(v)) \in P, \alpha \in \Sigma \cup N$	add $+2$ to H

Table 4.1: Rules for determining the harmony, H, of a parse tree, (V, E), for a string from an arbitrary context-free grammar, $(N \cup \Gamma, \Sigma, P, S)$. Here Γ is the set of new non-terminals that are added when constructing the normal form described in Section 3.4.1. l is a labelling of the nodes of the parse tree, $l : V \rightarrow \Sigma \cup N \cup \Gamma$.

that for every vertex in the parse tree, the number of incident edges that are required for the tree to be valid is known, because the grammar conforms to the normal form presented in Section 3.4.1. Thus for each vertex, the number of incident edges required is subtracted from the harmony. Each valid edge is incident to two vertices, and so, it contributes $+2$ to the harmony, balancing the negative contributions of the vertices at each end.

For example, let $F = (\{S, \Gamma_{1,1}, A, \Gamma_{2,1}, B, \Gamma_{3,1}, C, D\}, \{a, b\}, P, S)$ be a Context Free Grammar where P is the set of productions:

$$\begin{array}{lll} S \rightarrow \Gamma_{1,1} & \Gamma_{1,1} \rightarrow CB & A \rightarrow \Gamma_{3,1} \\ \Gamma_{3,1} \rightarrow CB & B \rightarrow \Gamma_{2,1} & B \rightarrow b \\ \Gamma_{2,1} \rightarrow AD & C \rightarrow a & D \rightarrow b \end{array}$$

Suppose $G = (V, E)$ is a potential derivation graph for a string in $L(F)$ and $l : V \rightarrow \{S, \Gamma_{1,1}, A, \Gamma_{2,1}, B, \Gamma_{3,1}, C, D, a, b\}$ is a labeling for the vertices of G. We can compute the harmony of G as follows:

- For any node v, if $l(v) \in \{a, b\}$ then add -1 to the harmony. (Rule 1 in Table 4.1)

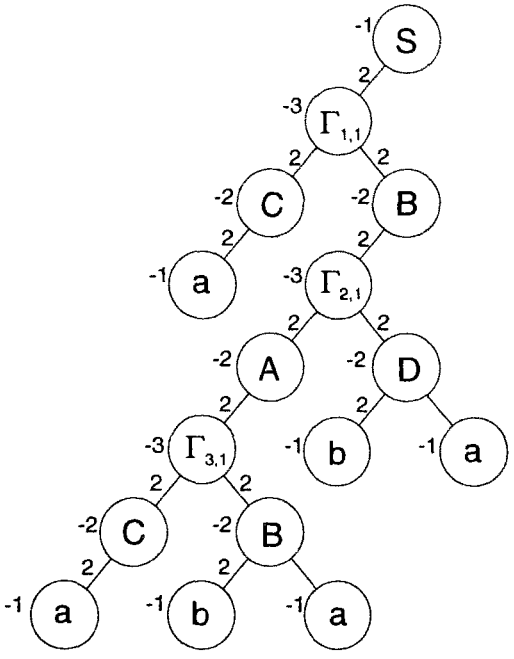

Figure 4.1: A parse tree for the word $w = aababa$ which is not in the language, $L(F)$ of the text. The numbers represent the contributions to the harmony value of the parse tree with respect to the grammar F of the text. It can be seen that the harmony of this tree is -2.

- For any node v, if $l(v) \in \{A, B, C, D, S\}$ then add -2 to the harmony. (Rule 2)
- If v is the root and $l(v) = S$ then add $+1$ to the harmony. (Rule 3)
- For any node v, if $l(v) \in \{\Gamma_{1,1}, \Gamma_{2,1}, \Gamma_{3,1}\}$ then add -3 to the harmony. (Rule 4)
- For any edge (u, v) where v is the first child of u, if $(l(u), l(v)) \in \{(S, \Gamma_{1,1}), (\Gamma_{1,1}, C), (A, \Gamma_{2,1}), (\Gamma_{2,1}, C), (B, \Gamma_{3,1}), (\Gamma_{3,1}, A), (C, a), (D, b), (B, b)\}$ then add 2 to the harmony. (Rule 5)
- For any edge (u, v) where v is the second child of u, if $(l(u), l(v)) \in \{(\Gamma_{1,1}, B), (\Gamma_{2,1}, B), (\Gamma_{3,1}, D)\}$ then add 2 to the harmony. (Rule 6)

Figures 4.1 and 4.2 illustrate this harmony function applied to an invalid and a valid parse tree, respectively.

The problem with this harmony function is that the method for constructing networks, discussed in Section 4.2 that find maximum harmony trees requires that

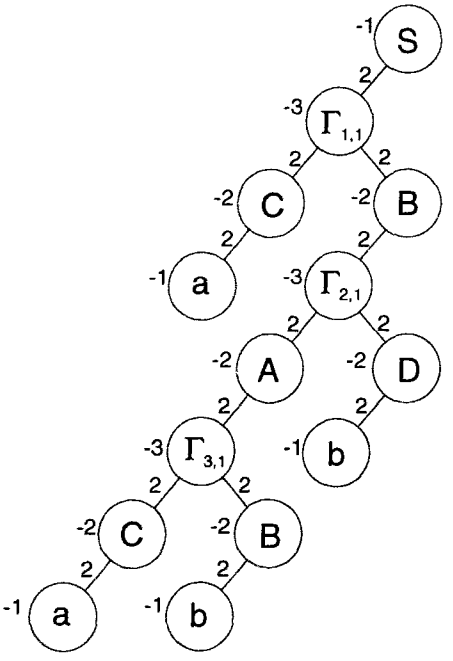

Figure 4.2: A parse tree for the word $w = aabb$ which is in the language, $L(F)$ of the text. The numbers represent the contributions to the harmony value of the parse tree with respect to the grammar F of the text. It can be seen that the harmony of this tree is 0.

the harmony function is *embedding invariant*. That is, the harmony function must determine the contribution to the harmony made by a pair of nodes without knowing where the pair lies in the overall structure of the tree. In the embedding invariant scheme, there is no way to tell if a particular node lies at the root of the tree, or below. This means that the rule that increments the harmony if the root is labelled with the start symbol cannot be used.

If the start symbol appears only on the left hand side of productions, a condition which is easily satisfied with a simple modification to the grammar, then the root is the only valid position for the start symbol. In that case, the start symbol will always contribute a value of -1 to the harmony value of the tree. Table 4.2 captures this modification and extends the rules for constructing a harmony function to apply to Type 0 grammars.

Production or Symbol	Change in Harmony
$l(v) = a \in \Sigma$	add -1 to H
$l(v) = A \in N \setminus \{S\}$	add -2 to H
$l(v) = S$	add -1 to H
$l(v) = \gamma \in \Gamma$ and $\{\alpha \to \gamma, \gamma \to \beta\} \subseteq P$, and $\alpha, \beta \in (N \cup \Sigma)$	add $-(\lvert\alpha\rvert + \lvert\beta\rvert)$ to H
$(u,v) \in E, u$ the 1st parent of $v \iff (l(u)\alpha \to l(v)) \in P$, or $(l(u) \to l(v)) \in P$ for some $\alpha \in (N \cup \Gamma \cup \Sigma)$	add $+2$ to H
$(u,v) \in E, u$ the 2nd parent of $v \iff (\gamma l(u) \to \gamma)) \in P$, for some $\gamma \in \Gamma$	add $+2$ to H
$(u,v) \in E, u$ the first child of $v \iff (l(u)\alpha \to l(v)) \in P$ or $(l(u) \to l(v)) \in P, \alpha \in \Sigma \cup N$	add $+2$ to H
$(u,v) \in E, u$ the second child of $v \iff (l(u) \to \alpha l(v)) \in P, \alpha \in \Sigma \cup N$	add $+2$ to H

Table 4.2: Rules for determining the harmony relative to an arbitrary grammar, G of a derivation graph (V, E) with vertex labelling given by l. The grammar is given by $G = (N \cup \Gamma, \Sigma, P, S)$ where Γ is the set of new non-terminals added when constructing the normal form as in Section 3.4.1.

4.1.2 Type 0 Derivation Graphs

Table 4.2 extends Smolensky, Legendre and Miyata's harmony function to a harmony function for an arbitrary type 0 grammar. This harmony function is simply an explicit version of the harmony function suggested by Smolensky, Legendre, and Miyata (1992). The proof is very similar to the development which was presented by Smolensky, Legendre, and Miyata (1994) and summarixed in the previous section, and so the harmony function is presented here without proof. As implied by the previous section, the revised harmony function not only gives an extension to type 0 grammars, but also makes the harmony function truly embedding invariant.

4.1.3 The Symbolic Viewpoint

The crucial aspect of both the original symbolic harmony function for Context Free parse trees and the variation presented above is that both calculate harmony by comparing only pairs of constituents. The harmony of the entire structure is calculated

by adding up the harmony values contributed by vertices and pairs of vertices (or vertices and edges). So, the harmony contribution of the edges can be expressed exactly as in the definition of the symbolic formulation of harmony:

$$H_{edges} = \sum_{i \neq j} H_{c_i r_i; c_j r_j}$$

The vertices, on the other hand, make a contribution regardless of their position. Thus,

$$H_{vertices} = \sum_{i} H_{c_i}$$

The vertex contributions can be included in the calculation of the edge contributions, but care should be taken so that each contribution is added exactly once. So, we can define the pairwise harmony,

$$\begin{aligned} H_{c_i r_i, c_j r_j} &= < \text{harmony contribution of edge between } r_i \text{ and } r_j > + \\ &\quad < \text{harmony contribution of } c_i > \end{aligned}$$

If c_i is the parent of c_j then we must ensure that there is a "null" child so that the sinks are also parents and their contributions are included in the sum. In this last case, we see that the harmony of the derivation graph can be expressed exactly as in Equation 4.1.

What's more, this formula for harmony is embedding invariant. That is the rules can be applied by looking at exactly two vertices, without any more information than their labels and the fact that one is the left or right parent of the other on a path from the source. This independence means that the roles r_i in the harmony calculation can be recursive — a property that will be useful as we turn to the numeric formulation of harmony theory.

4.2 Numeric Formulation

Smolensky (Smolensky, Legendre, and Miyata 1994) contends that symbolic computing is simply a high level view of human cognitive processes. From below, the processes are better viewed as the "spreading" of activation among connectionist units. If this

is true, then connectionists will have to devise a means to connect the two viewpoints of cognition: namely they will have to connect the symbolic formulation of harmony theory with a connectionist or numeric formulation.

Smolensky, Legendre and Miyata (Legendre, Miyata, and Smolensky 1990) note that with each state change, some types of neural networks minimize functions, called Liapunov functions, or energy functions. In particular, some neural networks serve to minimize the following function of their activation vector:

$$E(a) = -\frac{1}{2}a^T W a \tag{4.2}$$

Here a is the activation vector, and W is the neural network's weight matrix (Legendre, Miyata, and Smolensky 1990; Smolensky, Legendre, and Miyata 1992). Such a function will be called the *energy function* of the connectionist network as in (Cohen and Grossberg 1983; Hopfield 1982; Hopfield 1984; Hopfield 1987; Golden 1986; Salam, Wang, and Choi 1991; Li, Michel, and Porod 1988; Michel, Farrell, and Porod 1989; Lillo, Miller, Hui, and Zak 1994). Note that this terminology is different from that of Smolensky et al. who use the term "harmony function" to denote the negation of the energy function. To avoid confusion with the harmony function which has already been defined as a function mapping derivation graphs to the real numbers, we will use the more common term *energy function* for the function in equation 4.2. Subsection 4.2.1 will examine the networks that are known to minimize the above energy function.

Smolensky, Legendre and Miyata propose to exploit knowledge about energy functions to show that connectionist networks have the same computational powers as context free grammars. Their strategy is to describe a method of synthesizing the weight matrix for a connectionist network so that whenever the activation vector is a tensor representation of a parse tree, the energy of the network will be the same as the negation of the harmony of that parse tree (Smolensky, Legendre, and Miyata 1994, p154). Because valid parse trees have high harmony values, the low energy states of the network will be tensor representations of valid parse trees. Thus, by minimizing its own energy, such a network will naturally find representations of maximum harmony parse trees. From this point, Smolensky et al. argue that the connectionist

network exhibits the same ability as a context free grammar (Smolensky, Legendre, and Miyata 1992, p44).

The harmony proponents do not say how these trees are generated, nor indeed do they indicate how a network will go about calculating its own energy value and indicate its answer. These questions will be examined in detail in Chapter 5. Furthermore, they do not demonstrate that their harmony networks find maximum harmony parse trees — a question that is examined in Section 5.3. In this section we will simply trace the development of energy-minimizing connectionist networks which have the desired energy function. While it may be possible to synthesize such a network in an ad hoc manner, we desire a more structured approach.

In Section 4.1.3 we noted two principle attributes of the symbolic formulation which may help to construct these networks.

- For each pair of constituents, their roles are relative. In other words, the role of the second constituent can be expressed in terms of the first. For example, the harmony of a pair of nodes might be +2 if the second node is a valid child of the first.

- The harmony of a complete structure is the same as the sum of the harmonies of pairs of constituents in their roles.

Section 4.2.2 will examine the use of the first of these attributes to create a submatrix of the weight matrix. This submatrix will be used to calculate the negation of the harmony of a pair of adjacent vertices in the parse tree. Using this submatrix, Section 4.2.3 will examine the question of how copies of the submatrix can be combined to form the desired connectionist network — one for which the energy of tensor representations of parse trees is the negation of the harmony.

4.2.1 Connectionist Networks as Energy Minimization Systems

As noted previously, some connectionist networks, like a variety of other dynamic systems, admit a Liapunov or energy function. As the dynamic system evolves over

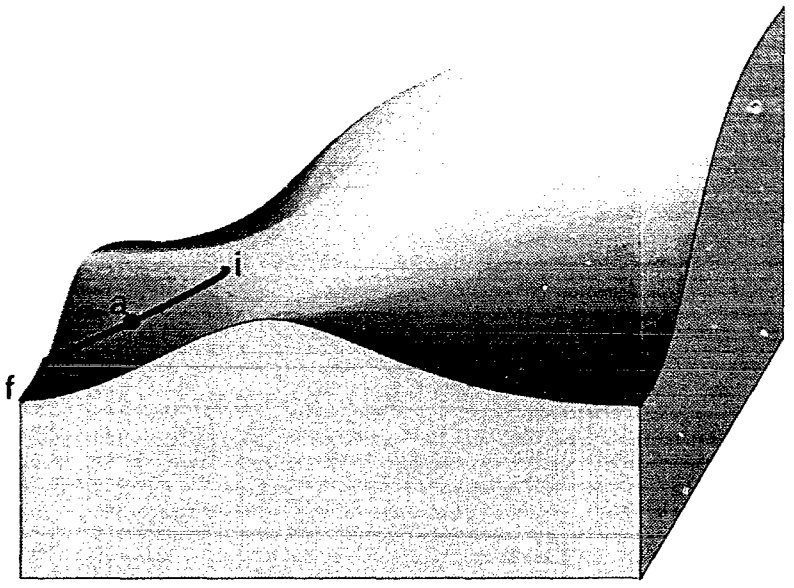

Figure 4.3: A graphical representation of a connectionist network's energy function. The activation vector, a, starts at an initial state i and moves "downhill" to a final equilibrium point, f.

time, the value of the Liapunov function decreases. In other words, the dynamic system seeks to minimize its Liapunov cost function. These functions ease the analysis of dynamic systems.

Informally, the energy function can be used to provide a visual metaphor for the operation of connectionist networks. Energy is a single valued function of a vector quantity. Thus, if the vector space is two dimensional, then the energy function can be viewed as a landscape, with hills and basins. The original state of the network is like a ball bearing placed randomly on the surface. As the connectionist network's state changes, the ball bearing falls toward the nearest basin until it comes to rest at an equilibrium point. Figure 4.3 represents this description graphically. For a reasonably understandable introduction to Liapunov functions see (Leipholz 1987), and for a description to these functions in the context of connectionism, see (Hopfield 1982). In these terms, Harmony theory seeks to devise a landscape so that the minima fall on the well-formed parse trees.

Harmony theory (Smolensky, Legendre, and Miyata 1992; Smolensky 1993) is based on the energy value, $E(a) = -\frac{1}{2}a^T W a$ of Equation 4.2. Smolensky et al. cite

papers by Cohen and Grossberg (1983), Golden (1986, 1988), Hinton, McClelland, and Rumelhart (1986), Hopfield (1982, 1984, 1987), and Smolensky (1986) to demonstrate that some connectionist models admit the above energy function. The models cited in these papers fall into special cases of two categories of connectionist models — the Brain State in a Box model (Golden 1986; Golden 1988), and the Hopfield model (Cohen and Grossberg 1983; Hinton, McClelland, and Rumelhart 1986; Hopfield 1982; Hopfield 1984; Hopfield 1987; Smolensky 1986).

Energy of the Brain-State-in-a-Box Model

The Brain-State-in-a-Box (BSB) Model, which is defined by Equations 2.3 and 2.4 on page 19, is one of the connectionist models that admits an energy function under certain conditions. This fact was proven by Golden (1986) using the following theorem:

Theorem 3 (Golden 1986) *If*

1. $E_{BSB}(a) = -\frac{1}{2}a^T W a$
2. W *is symmetric.*
3. *Either* $0 \leq \lambda_{min}$ *or* $\gamma < 2/|\lambda_{min}|$ *where* λ_{min} *is the minimum eigenvalue of* W*, and* γ *is the constant in Equation 2.3.*

Then

1. $E_{BSB}(a(t+1)) < E_{BSB}(a(t))$ *if* $a(t+1) \neq a(t)$.
2. $E_{BSB}(a(t+1)) = E_{BSB}(a(t))$ *if and only if* $a(t+1) = a(t)$.

This theorem states that $E_{BSB}(a)$ is monotonically non-increasing and that if it ever stops decreasing, then the BSB model comes to rest at a single point. Golden (1986) shows that these systems do in fact eventually come to rest at a single equilibrium point.

The hypothesis that stipulates W is symmetric corresponds to saying that activation travels equally well in each direction along the connections in the network. This

assumption is somewhat unusual from a practical standpoint because most implementations of connectionist networks have asymmetric weight matrices.

It is natural to ask whether the BSB must have a symmetric weight matrix in order to exhibit the desired energy function. In an experiment using 100 small (3 by 3) random asymmetric matrices, each admitted at least one initial vector that caused E_{BSB} to increase. For example, using the notation of Equations 2.3 and 2.4 let

$$W = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

If $\gamma = 0.1$ and $a(0) = (0.746, -1)^T$, then $E_{BSB}(a(0)) = -0.41$ $E_{BSB}(a(1)) = -0.395$, and $E_{BSB}(a(6)) = -0.35$. So the energy function admits an increase in this asymmetric case. Similarly, for those 100 asymmetric matrices, E_{BSB} is not a valid Liapunov function.

Energy of the Hopfield Model

The Hopfield model as defined in Equations 2.1 and 2.2 is quite different from the BSB model. Interestingly, under certain conditions it admits an energy function that is very similar to the energy function of the BSB model. Hopfield showed (Hopfield 1984), the following function is an energy function for Hopfield networks with symmetric weight matrices:

$$E_{Hopfield}(a) = -\frac{1}{2}a^T W a + \sum_i \frac{1}{R_i} \int_0^{a_i} g_i^{-1}(v)dv + \sum_i I_i a_i \tag{4.3}$$

Recall from Section 2.6 that R_i is a biologically motivated constant, g_i is the response curve of unit i, and I_i is a fixed input signal to the unit, i.

Yang and Dillon (1994) modified this energy function and demonstrated an instability result to show that the modification was necessary. They assert that the reason $E_{Hopfield}(a)$ is inadequate is that it assumes that the network will reach an asymptotically stable equilibrium when the energy function is minimized. Their instability result shows that this is not necessarily true — the state vector can oscillate between two equilibria of equal energy. The modification defines an equilibrium as a point

where there is no change in the activation vector. They then go on to modify the energy function, Equation 4.3, to measure the distance in terms of $E_{Hopfield}(a)$ between an equilibrium point and the activation vector, a. Since $E_{Hopfield}(a)$ is much simpler than the energy equation proposed by Yang et al., the remainder of this thesis will make the reasonable assumption that the minima are indeed asymptotically stable equilibria, in which case, $E_{Hopfield}(a)$ is adequate to denote the energy function of the Hopfield network.

In (Hopfield 1984), Hopfield showed that under certain conditions, the Hopfield model has the same energy function as that of the Brain-State-in-a-Box model. In the case where all the units are allowed to vary freely, from their initial state, I_i is zero for all i, and so the last term is zero. Recalling that $g_i(v)$ is an odd function, with $|g_i(v)| \leq 1$, the integral in the second term also becomes zero as $g_i(v)$ becomes steeper. Thus when all the units are allowed to vary freely, and the units' response curves ($g_i(v)$) are steep, and the weight matrix is symmetric, the energy function of the symmetric Hopfield model reduces to:

$$E_{Hopfield}(a) = -\frac{1}{2}a^T W a = E_{BSB}(a) \tag{4.4}$$

So, given these conditions, the symmetric Hopfield model has the same energy function as the symmetric brain-state-in-a-box model.

4.2.2 Pairwise Harmony in Context Free Parse Trees

We will use the energy function described in the previous section to provide insight into how a connectionist network can be constructed so that its energy function is the negation of the harmony function for some grammar. As indicated at the beginning of Section 4.2, we will exploit the fact that the change in harmony attributed to a pair of vertices is non-zero only if one vertex is the parent of the other.

Recall that in a recursive tensor representation, the representation is simply the sum of the tensor products of the symbols with their recursive roles. In the cases where the recursive roles differ in dimension, the sum is taken to be a direct sum. Thus if the activation vector a is a superposition vector containing some recursive

For every ...	Add equations ...
terminal, f_1	$-\frac{1}{2}(f_1+\vec{0}\otimes r_l)^T W_{root}(f_1+\vec{0}\otimes r_l) = +1$
non-terminal, $f_1 \in N \setminus \{S\}$	$-\frac{1}{2}(f_1+\vec{0}\otimes r_l)^T W_{root}(f_1+\vec{0}\otimes r_l) = +2$
non-terminal, $f_1 \in \Gamma$	$-\frac{1}{2}(f_1+\vec{0}\otimes r_l)^T W_{root}(f_1+\vec{0}\otimes r_l) = +3$
non-terminal $f_1 = S$	$-\frac{1}{2}(f_1+\vec{0}\otimes r_l)^T W_{root}(f_1+\vec{0}\otimes r_l) = +1$
production $f_1 \rightarrow f_2 f_3$ where $f_3 \in N \cup \Gamma \cup \Sigma \cup \{\epsilon\}$	$-\frac{1}{2}(f_1+\vec{0}\otimes r_l)^T W_{root}(\vec{0}+f_2\otimes r_l) = -1$ $-\frac{1}{2}(\vec{0}+f_2\otimes r_l)^T W_{root}(f_1+\vec{0}\otimes r_l) = -1$
production $f_1 \rightarrow f_2 f_3$ where $f_3 \in N \cup \Gamma \cup \Sigma$	$-\frac{1}{2}(f_1+\vec{0}\otimes r_l)^T W_{root}(\vec{0}+f_3\otimes r_r) = -1$ $-\frac{1}{2}(\vec{0}+f_3\otimes r_r)^T W_{root}(f_1+\vec{0}\otimes r_r) = -1$
i, j	$(W_{root})_{i,j} = (W_{root})_{j,i}$

Table 4.3: The system of equations that determines W_{root} for a context free grammar, $(N \cup \Gamma, \Sigma, P, S)$. This system of equations is extended from that proposed by Smolensky, Legendre, and Miyata (1992). Note that we are abusing the notation by using f_i to denote both symbols in the grammar, and vector representations of those symbols. $\vec{0}$ is a zero vector in the same space as $f_1, \ldots, f_3$. As in Chapter 3, r_l and r_r are role vectors denoting left child and right child respectively.

tensor representation, then

$$a = \sum_i f_i \otimes r_i$$

If each of the summands is taken to be a vector in the direct sum then they could be added as vectors. That is, if we pad the vector $f_i \otimes r_i$ with an appropriate number of zeros so that the dimensions are consistent, we can express the superposition vector as

$$a = \sum_i (\vec{0_{i_1}}, f_i \otimes r_i, \vec{0_{i_2}})$$

Here, all the summands lie in the same vector space and can be added.

If the connectionist network is one of those described in the previous section, and if the activation vector is interpreted as a parse tree, then, the energy of the network will be

$$E(a) = -\frac{1}{2} a^T W a$$

$$= -\frac{1}{2}\left(\sum_i(\vec{0_{i_1}}, f_i \otimes r_i, \vec{0_{i_2}})^T\right) W \left(\sum_j(\vec{0_{j_1}}, f_j \otimes r_j, \vec{0_{j_2}})\right)$$
$$= -\frac{1}{2}\sum_i\sum_j(\vec{0_{i_1}}, f_i \otimes r_i, \vec{0_{i_2}})^T W(\vec{0_{j_1}}, f_j \otimes r_j, \vec{0_{j_2}})$$

This equation is very similar to the symbolic formulation of harmony that is presented in Section 4.1. Like the symbolic harmony function, the weight matrix relates pairs of constituents.

The recursive representation of parse trees allows further characterization of the weight matrix W. When calculating the harmony of a parse tree, the contribution of a pair of vertices is non-zero only when the pair is connected by an arc. We are only interested, then, in two symbols if one appears as either a left-child or a right-child of the other. If r_l and r_r denote the left- and right-child roles, respectively, then in order for the harmony contribution of two symbols to be non-zero, one symbol must appear in a role which is the tensor product of the second role and either r_l or r_r. In other words, $(\vec{0_1}, f_1 \otimes r_1, \vec{0_2})^T W(\vec{0_3}, f_2 \otimes r_2, \vec{0_4}) \neq 0$ only if $r_2 = r \otimes r_1$ or $r_1 = r \otimes r_2$ for some $r \in \{r_l, r_r\}$.

Because the roles of two adjacent vertices are related in this way, the two constituents appear in adjacent locations in the direct sum of all the constituents. In other words, whenever the harmony contribution of a pair of vertices is non-zero, the padding vectors, $\vec{0_1}$ and $\vec{0_3}$ are "nearly" the same, as are $\vec{0_2}$ and $\vec{0_4}$. So, the non-zero entries of the weight matrix occur in a relatively small section "near" the main diagonal.

What's more, as discussed in Section 4.1.3, the harmony function for context free parse trees is independent of embedding. That is, the harmony contribution of a maximal subtree of the parse tree is the same regardless of where it occurs in the whole tree. Consequently, the connections that relate two constituents will be the same regardless of how deeply the constituents are embedded. Therefore, from the whole weight matrix, we need concern ourselves for now only with a small submatrix that relates two constituents. Smolensky et al. call this submatrix W_{root}.

It is reasonably straightforward to find a value for W_{root} so that the energy contributions of two constituents are equal to the negation of their harmony values. Finding

W_{root} is merely a matter of solving a system of equations which is given in Table 4.3. If the vector representations of the terminal and non-terminal symbols are linearly independent, then this system of equations will always have at least one solution. There may also be a solution if the vectors representing the symbols are linearly dependent.

Consider for example the regular grammar, $F = (N \cup \Gamma, \Sigma, P, S)$ where the productions are

$$S \to a \quad S \to \Gamma_{1,1} \quad \Gamma_{1,1} \to aB$$
$$B \to a \quad B \to \Gamma_{1,1}$$

This grammar is in the normal form of Section 3.4.1. We can see that W_{root} must satisfy the following equations:

$$\begin{aligned}
(S + \vec{0} \otimes r_l)^T W_{root}(S + \vec{0} \otimes r_l) &= -2 \\
(\Gamma_{1,1} + \vec{0} \otimes r_l)^T W_{root}(\Gamma_{1,1} + \vec{0} \otimes r_l) &= -6 \\
(B + \vec{0} \otimes r_l)^T W_{root}(B + \vec{0} \otimes r_l) &= -4 \\
(a + \vec{0} \otimes r_l)^T W_{root}(a + \vec{0} \otimes r_l) &= -2 \\
(S + \vec{0} \otimes r_l)^T W_{root}(\vec{0} + a \otimes r_l) &= 2 \\
(\vec{0} + a \otimes r_l)^T W_{root}(S + \vec{0} \otimes r_l) &= 2 \\
(S + \vec{0} \otimes r_l)^T W_{root}(\vec{0} + \Gamma_{1,1} \otimes r_l) &= 2 \\
(\vec{0} + \Gamma_{1,1} \otimes r_l)^T W_{root}(S + \vec{0} \otimes r_l) &= 2 \\
(\Gamma_{1,1} + \vec{0} \otimes r_l)^T W_{root}(\vec{0} + a \otimes r_l) &= 2 \\
(\vec{0} + a \otimes r_l)^T W_{root}(\Gamma_{1,1} + \vec{0} \otimes r_l) &= 2 \\
(\Gamma_{1,1} + \vec{0} \otimes r_l)^T W_{root}(\vec{0} + B \otimes r_r) &= 2 \\
(\vec{0} + B \otimes r_l)^T W_{root}(\Gamma_{1,1} + \vec{0} \otimes r_l) &= 2 \\
(B + \vec{0} \otimes r_l)^T W_{root}(\vec{0} + a \otimes r_l) &= 2 \\
(\vec{0} + a \otimes r_l)^T W_{root}(B + \vec{0} \otimes r_l) &= 2 \\
(B + \vec{0} \otimes r_l)^T W_{root}(\vec{0} + \Gamma_{1,1} \otimes r_l) &= 2 \\
(\vec{0} + \Gamma_{1,1} \otimes r_l)^T W_{root}(B + \vec{0} \otimes r_l) &= 2 \\
W_{i,j} &= W_{j,i}
\end{aligned}$$

We will use the following vector representations for the terminal and non-terminal

symbols:

$$S = (1,0,0,0) \quad \Gamma_{1,1} = (0,1,0,0)$$
$$B = (0,0,1,0) \quad a = (0,0,0,1)$$

Also, we will use the role vectors $r_l = (1,0)$ and $r_r = (0,1)$. Substituting these values into the equations above results in the following value for W_{root}:

$$W_{root} = \left(\begin{array}{cccc|cccccccc} -2 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 2 & 0 \\ 0 & -6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 2 & 0 \\ 0 & 0 & -4 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

That this value for W_{root} yields the correct energy values is easily checked.

Due to the nature of the equations that define W_{root}, it admits four distinct regions. If symbols are vectors of dimension m and roles are n dimensional, then we can write

$$W_{root} = \begin{pmatrix} A & B \\ B^T & D \end{pmatrix}$$

Where

A is a $m \times m$ block which is used to calculate the contribution to energy of the root of the pair by itself.

B is a $m \times mn$ block which is used to calculate half the contribution to the energy of edge between the vertices.

B^T is a $mn \times m$ block which is used to calculate half the contribution to the energy of edge between the vertices.

D is a $mn \times mn$ block of unconstrained variables which we will take to be zeros.

So we now have a way to calculate the harmony of two constituents represented in tensor form.

4.2.3 Harmony of Whole Context Free Parse Trees

In principle, the matrix, W_{root} which relates any two constituents should be "embedded" at every level of the weight matrix. In this way, the harmony of any two adjacent vertices of the parse tree can be calculated, regardless of their embedding. Due to the recursive nature of the tensor representation for parse trees, Smolensky et al. argue that the correct way to perform this embedding is with the recursion formula

$$\begin{aligned} W &= W_{root} + I \otimes W \\ &= W_{root} + I \otimes W_{root} + I \otimes I \otimes W_{root} + \ldots \end{aligned}$$

It is difficult to know what exactly they mean by this formula. W is the symmetric weight matrix for a connectionist network, and W_{root} is the submatrix developed above that relates two constituents. I is surely the identity matrix with the same dimensions as the role vectors. But, then the summands on the right hand side all have different dimensions. They are added together as elements of the direct sum, so the right hand

side of this equation is essentially a vector, not a matrix. For example, if I is 2×2 it will be:

$$W = \begin{pmatrix} W_{root} & & & \\ & \begin{pmatrix} W_{root} & 0 \\ 0 & W_{root} \end{pmatrix} & & \\ & & \begin{pmatrix} W_{root} & 0 & 0 & 0 \\ 0 & W_{root} & 0 & 0 \\ 0 & 0 & W_{root} & 0 \\ 0 & 0 & 0 & W_{root} \end{pmatrix} & \\ & & & \ddots \end{pmatrix}$$

Moreover, left-multiplying by the identity matrix does not change the size of the non-zero blocks in the weight matrix. However, the dimension of a parent-child pair in the tensor representation of the parse tree depends on the level of embedding, and can be arbitrarily large.

Fortunately, it is not hard to see how to properly construct W. Note that if symbols are vectors of dimension m and roles have dimension n, then

$$W_{root} = \begin{pmatrix} A_{m\times m} & B_{m\times mn} \\ B^T_{mn\times m} & D_{mn\times mn} \end{pmatrix}$$

The block W_{root} can be expanded by right-multiplying by a matrix. For example, if $I_{n\times n}$ is the $n \times n$ identity matrix, then

$$W_{root} \otimes I_{n\times n} = \begin{pmatrix} (A \otimes I)_{mn\times mn} & (B \otimes I)_{mn\times mn^2} \\ (B^T \otimes I)_{mn^2\times mn} & (D \otimes I)_{mn^2\times mn^2} \end{pmatrix}$$

Because D is always zero and it has the same dimensions as $A \otimes I$, we can replace D with $A \otimes I$. Then the weight matrix, W would be given by

$$W = \begin{pmatrix} A & B & 0 & 0 & \cdots \\ C & A \otimes I & B \otimes I & 0 & \cdots \\ 0 & C \otimes I & A \otimes I \otimes I & B \otimes I \otimes I & \\ 0 & 0 & C \otimes I \otimes I & A \otimes I \otimes I \otimes I & \\ \vdots & \vdots & & & \ddots \end{pmatrix} \tag{4.5}$$

That the energy function for a connectionist network with this weight matrix is the negation of the harmony is proven below by Theorem 4.

Before proving that theorem, however, we will need two small lemmas. The first lemma states that the value $a^T W b$ remains the same if both a and b are tensor-multiplied by the same unit-length role vector. The second lemma shows that if a and b are tensor-multiplied by different orthogonal role vectors, then the value $a^T W b$ is annihilated. The theorem uses these two lemmas to show that if s is the tensor representation of a tree, then $-\frac{1}{2} s^T W s$ is the same as the harmony of that tree.

Lemma 1 *If*

1. *W is as defined in Equation 4.5.*
2. *r is a role vector, $r^T = (r_1, r_2, \ldots)$.*
3. *$r^T r = 1$*
4. *$\vec{0}$ is a zero vector in the space of symbol vectors*
5. *$a^T = (a_1, a_2, \ldots)$, and $b^T = (b_1, b_2, \ldots)$*

Then

$$(\vec{0}^T, (a \otimes r)^T) W \begin{pmatrix} \vec{0} \\ b \otimes r \end{pmatrix} = a^T W b$$

Proof:

$$\begin{aligned}
& (\vec{0}^T, (a \otimes r)^T) W \begin{pmatrix} \vec{0} \\ b \otimes r \end{pmatrix} \\
= \; & (a \otimes r)^T (W \otimes I)(b \otimes r) \\
= \; & (a_1 r^T, a_2 r^T, \ldots) \begin{pmatrix} W_{1,1} I & W_{1,2} I & \cdots \\ W_{2,1} I & W_{2,2} I & \\ \vdots & & \ddots \end{pmatrix} \begin{pmatrix} b_1 r \\ b_2 r \\ \vdots \end{pmatrix} \\
= \; & \left(\sum_i a_i W_{i,1} r_1, \sum_i a_i W_{i,1} r_2, \ldots, \sum_i a_i W_{i,2} r_1, \ldots \right) \begin{pmatrix} b_1 r \\ b_2 r \\ \vdots \end{pmatrix}
\end{aligned}$$

$$
\begin{aligned}
&= \left(\left(\sum_i a_i W_{i,1}\right) r^T, \left(\sum_i a_i W_{i,2}\right) r^T, \ldots\right) \begin{pmatrix} b_1 r \\ b_2 r \\ \vdots \end{pmatrix} \\
&= \sum_j \left(b_j \left(\sum_i a_i W_{i,j}\right) r^T r\right) \\
&= \sum_i \sum_j a_i W_{i,j} b_j \\
&= a^T W b \square
\end{aligned}
$$

Lemma 2 *If*

1. *W is as defined in Equation 4.5.*
2. *r_1, r_2 are role vectors such that $r_1^T r_2 = 0$ and $r_1^T = (r_{1,1}, r_{1,2}, \ldots), r_2^T = (r_{2,1}, r_{2,2}, \ldots)$.*
3. *$\vec{0}$ is a zero vector in the space of symbol vectors*
4. *$a^T = (a_1, a_2, \ldots)$, and $b^T = (b_1, b_2, \ldots)$*

Then

$$(\vec{0}^T, (a \otimes r_1)^T) W \begin{pmatrix} \vec{0} \\ b \otimes r_2 \end{pmatrix} = 0$$

Proof:

$$
\begin{aligned}
& (\vec{0}^T, (a \otimes r_1)^T) W \begin{pmatrix} \vec{0} \\ b \otimes r_2 \end{pmatrix} \\
= \; & (a \otimes r_1)^T (W \otimes I)(b \otimes r_2) \\
= \; & (a_1 r_1^T, a_2 r_1^T, \ldots) \begin{pmatrix} W_{1,1} I & W_{1,2} I & \cdots \\ W_{2,1} I & W_{2\,2} I & \\ \vdots & & \ddots \end{pmatrix} \begin{pmatrix} b_1 r_2 \\ b_2 r_2 \\ \vdots \end{pmatrix} \\
= \; & \left(\sum_i a_i W_{i,1} r_{1,1}, \sum_i a_i W_{i,1} r_{1,2}, \ldots, \sum_i a_i W_{i,2} r_{1,1}, \ldots\right) \begin{pmatrix} b_1 r_2 \\ b_2 r_2 \\ \vdots \end{pmatrix}
\end{aligned}
$$

$$
\begin{aligned}
&= \left(\left(\sum_i a_i W_{i,1}\right) r_1^T, \left(\sum_i a_i W_{i,2}\right) r_1^T, \ldots\right) \begin{pmatrix} b_1 r_2 \\ b_2 r_2 \\ \vdots \end{pmatrix} \\
&= \sum_j \left(b_j \left(\sum_i a_i W_{i,j}\right) r_1^T r_2\right) \\
&= 0\square
\end{aligned}
$$

Theorem 4 *If*

1. W_{root} *is as defined in Section 4.2.2.*
2. $H_{f_1 r_1; f_2 r_2}$ *is the contribution to the harmony of the pair of vertices labelled* f_1, f_2 *in the roles* r_1, r_2 *as is Section 4.1.3*
3. $H_{f,r}$ *is the harmony contribution of a vertex labelled* f *in the role* r.
4. W *is as defined in 4.5.*
5. $s = \sum_i f_i \otimes r_i$ *is a tensor representation of a tree as in Section 3.3.*

Then

$$s^T W s = -2\left(\sum_{i \neq j} H_{f_i r_i; f_j r_j} + \sum_i f_i, r_i\right)$$

Proof: *(by structural induction on s)*

If $s = 0$ *(representing a tree with no vertices), then* $s^T W s = 0$.

If $s = f$ *(representing a tree with a single vertex which is labelled* f*), then because of the definition of* W_{root},

$$
\begin{aligned}
s^T W s &= (f + \vec{0} \oslash r_l)^T W_{root} (f + \vec{0} \oslash r_l) \\
&= -2 H_{f, root}
\end{aligned}
$$

Let $s_l = \sum_i f_{l,i} \otimes r_{l,i}$ *and* $s_r = \sum_i f_{r,i} \otimes r_{r,i}$ *be tensor represeniations for trees as in Section 3.3. Let* $f_{l,1}, f_{r,1}$ *be the labels of the roots of their respective trees. Assume*

that

$$\begin{aligned} s_l^T W s_l &= -2\left(\sum_{i\neq j} H_{f_{l,i}r_{l,i};f_{l,j}r_{l,j}} + \sum_i f_{l,i}, r_{l,i}\right) \\ s_r^T W s_r &= -2\left(\sum_{i\neq j} H_{f_{r,i}r_{r,i};f_{r,j}r_{r,j}} + \sum_i f_{r,i}, r_{r,i}\right) \end{aligned}$$

Then $s = f_1 + s_l \oslash r_l + s_r \oslash r_r$ *is a tensor representation for a tree as in Section 3.3. Now, because* W *is symmetric,*

$$\begin{aligned} s^T W s &= f_1^T W f_1 + f_1^T W(\vec{0} s_l \oslash r_l) + f_1^T W(\vec{0} + s_r \otimes r_r) + \\ &\quad (\vec{0} + s_l \oslash r_l)^T W f_1 + (\vec{0} + s_l \otimes r_l)^T W(\vec{0} + s_l \otimes r_l) + \\ &\quad (\vec{0} + s_l \oslash r_l)^T W(\vec{0} + s_r \oslash r_r) + (\vec{0} + s_r \otimes r_r)^T W f_1 + \\ &\quad (\vec{0} + s_r \oslash r_r)^T W(\vec{0} + s_l \oslash r_l) + (\vec{0} + s_r \otimes r_r)^T W(\vec{0} + s_r \otimes r_r) \\ &= f_1^T W f_1 + 2 f_1^T W(\vec{0} s_l \otimes r_l) + 2 f_1^T W(\vec{0} + s_r \otimes r_r) + \\ &\quad (\vec{0} + s_l \otimes r_l)^T W(\vec{0} + s_l \oslash r_l) + 2(\vec{0} + s_l \otimes r_l)^T W(\vec{0} + s_r \otimes r_r) + \\ &\quad (\vec{0} + s_r \oslash r_r)^T W(\vec{0} + s_r \oslash r_r) \end{aligned}$$

With lemmas 1 and 2, it can be seen that this is simply

$$\begin{aligned} s^T W s &= f_1^T W f_1 + 2 f_1^T W(\vec{0} + s_l \otimes r_l) + 2 f_1^T W(\vec{0} + s_r \otimes r_r) + \\ &\quad s_l^T W s_l + 0 + s_r^T W s_r \end{aligned}$$

Because $W_{i,j} = 0$ ***for*** $1 \leq i \leq m, (mn + m) \leq j$, ***the only contribution to the first, second and third terms comes from the interaction through*** W_{root} ***and so,***

$$\begin{aligned} s^T W s &= f_1^T W_{root} f_1 + 2 f_1^T W_{root}(\vec{0} + f_{l,1} \otimes r_l) + \\ &\quad 2 f_1^T W_{root}(\vec{0} + f_{r,1} \otimes r_r) + s_l^T W s_l + \\ &\quad s_r^T W s_r \end{aligned}$$

Finally, by the induction hypothesis, and the construction of W_{root}, *we see that*

$$\begin{aligned} s^T W s &= -2H_{f_1 root} + -2H_{f_1 root; f_{l,1} r_l} + -2H_{f_1 root; f_{r,1} r_r} + \\ &\quad -2\left(\sum_{i \neq j} H_{f_{l,i} r_{l,i}; f_{l,j} r_{l,j}} + \sum_i f_{l,i}, r_{l,i}\right) + \\ &\quad -2\left(\sum_{i \neq j} H_{f_{r,i} r_{r,i}; f_{r,j} r_{r,j}} + \sum_i f_{r,i}, r_{r,i}\right) \end{aligned}$$

Which is as required. □

So, the formula for W in Equation 4.5 results in a connectionist network whose energy values are the same as the negation of the harmony of the activation vector taken as a tensor representation for a tree. A connectionist network that minimizes the required energy function, and which has the above weight matrix relative to a particular grammar, G, will be called a *harmony network* for G.

Surely, all that remains now is to construct such a harmony network and it will compute maximum-harmony trees. Before constructing such a network and analyzing its behavior, however, we will pause to consider two issues that have been buried under the details of the past two chapters. First, we will discuss, in Section 5.1 whether it is meaningful to have a network which computes maximum-harmony trees. We will then go on to look at the size of these networks in Section 5.2. Finally, in Section 5.3, we will look at harmony networks, and determine if they do in fact compute maximum-harmony trees.

Chapter 5

Discord

5.1 Input and Output in Harmony Networks

As suggested in Chapter 1, we are interested in connectionist networks that have the same computational power as certain classes of machines or classical algorithms. This section examines the harmony networks constructed in the previous chapter and compares them to the Turing Machine (TM) model of computation. While Smolensky, Legendre and Miyata do claim Harmony networks can accept arbitrary formal languages (Smolensky, Legendre, and Miyata 1992), such power may be unnecessary to simulate cognitive processes. However, the portions of the TM model used in this section could easily be replaced with similar parts of any other machine model of computation.

Recall that a TM for a recursive language L consists of a finite control and an infinite tape. In the initial state, the tape contains the input, or a string to be tested for membership in L. The processor scans the tape, erasing symbols and writing some symbols of its own. In its final state, the tape contains a single symbol indicating that the machine either accepted or rejected the input.

A harmony network for a grammar, G, by contrast, consists of a set of units connected by weighted edges. The harmony proponents do not propose a means by which the input string can be presented to the network, and so the activation values in the initial state are undefined. The harmony network proceeds to relax into a

minimum-energy state where the activation values represent a maximum-harmony parse tree relative to G. This final state is related to acceptance or rejection by calculating the harmony of the parse tree, or equally by determining the energy of the network in its final, stable state.

There is no direct correspondence between either the initial states or the final states of these two models. Sections 5.1.1 and 5.1.2 will examine each of these discrepancies in turn. This scrutiny will be followed in Section 5.1.3 by a proposed definition for connectionist model of computation that is comparable to the Turing machine model. Section 5.1.4 will relate the decision networks of Section 5.1.3 to harmony theory in order to define harmonic decision networks. Finally, Section 5.1.5 will investigate a special class of harmonic decision networks.

5.1.1 Input

So far, the initial state of a harmony network is undefined. As suggested above, in order to correspond with the TM model, this initial state must involve, in some way, the word that is to be tested for membership in the language. We will examine three different schemes by which the initial state can include a representation of the word. None of these strategies is found to be satisfactory.

One way to present the word to the system is to make the initial state a representation of the word. For example, various groups of units could be used to represent each symbol in the input string. However, the structure of a harmony network does not guarantee that the parse tree representation that results from a particular word being present in the initial state will have anything to do with that word.

To be a parse tree for the word represented by the initial state, the final state must be a local minimum for a region (called a "basin" in Section 4.2.1) that includes the initial state. The harmony network and energy function was not constructed with this in mind, and so, it is extremely unlikely to happen. In particular, it could be the case that a harmony network could settle into a representation of a parse tree for a word that is entirely different from the word presented as input. The final state will not be related to the input word, and so, the calculation will not yield the desired

result — a resolution of the word's membership in the language.

Smolensky (1993) proposes that the network completes a representation when provided with a partial representation. Indeed, experiments show that many connectionist networks of the type proposed by harmony theory tend to function in that way. This seems to indicate that, given part of a parse tree for a given word, the network will complete the tree and then calculate the energy value to determine if the tree is valid or not. The problem is this: we cannot specify the word without giving the form of the entire parse tree, thus defeating the purpose of building a network that can supposedly find valid parse trees.

Recall that the tensor representation of a parse tree (see Section 3.3) is constructed recursively by adding the tensor products of the left- and right-child roles with the representations of the left and right subtrees to the representation for the root. Because the particular product that is considered in tensor representations distributes over addition, this construction amounts to adding all the labels of the vertices each in their own recursive role. These recursive roles specify a simple path from each vertex to the root of the tree.

The only part of the parse tree that is known in the initial state is the word itself, which constitutes the leaves of the parse tree. As long as the empty string is not in the language, we can assume, without loss of generality, that the grammar has no productions of the form, $A \rightarrow \epsilon$; so the input word accounts for every leaf in the parse tree.

Therefore, to specify the symbols in the word in their final positions, we must specify a path from every leaf to the root of the tree. Thus, specifying the terminal symbols and their positions amounts to providing a description of the whole tree, albeit with the vertices left unlabelled. So, if we can construct a parse tree for a word, in the language, then the harmony network will label the vertices and perhaps answer that the word is in the language.

There are two ways that we could determine such a parse tree — by parsing the word, or randomly. If the former course is followed, then the parse tree's labelling as well as the word's membership in the language would already be discovered prior to using the network. That defeats the purpose of building the network in the first place.

If, on the other hand, the parse tree is generated randomly, the network would have to try to label every appropriate binary tree before it could answer definitively that the word is not in the language. Because there are in principle an infinite number of unlabelled binary trees with a given sequence of leaves, this network would have to consume infinite resources before it could conclude that the word was not in the language. Therefore, we see that providing the leaves of the parse tree as the initial state is not reasonable.

Another proposal for the presentation of the input is to use it to constrain the state space (Legendre, Miyata, and Smolensky 1990). The idea here is that the activation vector is a triple, (i, h, o) where i is the input set, o is the output set and h is the harmony network. Normally, the activation vector can vary throughout its entire state space. Fixing the values of the input units is equivalent to constraining the activation vector to a smaller part of its state space.

To calculate the parse tree of a particular word, the input units are held constant so they represent the word. The network is then allowed to find its minimum-energy state in the constrained state space. The resulting minimum-energy state could be the required parse tree.

However, the structure of the harmony network does not guarantee that the location of the parse tree for a word will fall in a particular region of the state space. Indeed, the construction of the harmony network does not even mention the input units at all, let alone indicate how to connect them to the rest of the network so that they confine the state space appropriately. So, the stable equilibrium that results from constraining the inputs could just as likely be a parse tree for a different word than that presented to the input units.

5.1.2 Output

In order for harmony networks to closely match the Turing machine model, there must be a way to determine the output. Ideally, we would like the output to be a single unit which has two possible final values — one denoting acceptance and the other denoting rejection. Now, the final state of the harmony network is a tensor

representation of a parse tree. Because each unit in the parse tree representation is used in the representation of one or more symbols in positions within the tree, there is no single unit which we can examine to determine if the network has found a valid parse tree or not.

The factor that determines whether or not the parse tree is valid is the harmony of the overall structure. Equivalently, it is the energy of the harmony network. If the energy is zero in the final state, then the word represented at the leaves is in the language, otherwise it should be rejected.

Unfortunately, while they implicitly minimize energy, harmony networks do not possess any way to explicitly calculate that value. Harmony networks were constructed so that their final state would be a representation of a valid parse tree, but no part of the network was set aside to calculate the harmony of the rest of the network, and report whether a valid or an invalid parse tree was found. At a stable equilibrium, the harmony network can only say that it has reached the lowest energy point attainable from its initial state. It does not have any knowledge about the value of the energy at that point.

If the harmony network itself cannot determine its own energy, perhaps an external process could examine the network and determine the energy value. Naturally, we would want the external process to be a connectionist system as well. We will call this external process the *energy calculator.*

Now, in order to calculate the energy value of the harmony network, the energy calculator must perform the following sum:

$$\begin{aligned} E(a) &= -\frac{1}{2}a^T W a \\ &= \sum_i \sum_j a_i W_{i,j} a_j \end{aligned}$$

To determine each summand, the energy calculator must have access not only to the activation values of the nodes in the harmony network, but the weights on the links between nodes. The activation values are not a problem because they can simply be "read off"; however, the fact that the energy calculator must know the weights of the links between nodes seems to contradict one of the basic tenets of connectionism — that all calculation must be local, and all signals are simple numbers.

Because they can't calculate their own energy, and no connectionist energy calculator can calculate it for them, harmony networks have no way of knowing if the word present in the initial state is a member of the language or not. A harmony network may find a final state which has the required energy value to distinguish the activation vector as representing a valid parse tree. However, the network will keep this energy value, and the resulting knowledge to itself.

In order to be useful, the connectionist network should have some designated set of output units. To determine if the word presented on the input units was in the language, the output units would then be examined to see if their pattern of activation indicated "accept" or "reject." In the following discussion, we will consider networks which have a designated set of input units and a single output unit in more detail. We will call such networks *decision networks.*

5.1.3 Decision Networks

To decide if a word is in the language, a representation of that word must be presented on the decision network's input units. In some connectionist models, each unit can have only a finite number of different states. Because the network has only a finite number of input units, and each of these may have only a finite number of possible states, such models can only accept finite languages, and regular languages (see below).

Unfortunately most interesting languages are infinite, and so, most languages could not be accepted by a decision network. Notice that this concern is not unique to connectionist decision networks: all implementations of Turing machines are also finite, and thus can only accept strings of finite length. In practice, then any particular implementation of a Turing machine can only accept a finite language, and so it will be with decision networks:

Definition 1 *Let $N = (U, E)$ be a connectionist network, where $I \subset U$ is the set of input units which vary throughout the space $D^{|I|}$ where D is a finite subset of $\mathbf{R}$. Let d_{min} and d_{max} be the minimum and maximum elements of D respectively. Let $\Sigma = \{d_{min}, d_{max}\}$ be an alphabet and let $L \subseteq \Sigma^{\leq n}$ be a language over that alphabet*

whose strings all have length at most n. Let a_{out} be the value of the single output unit. Then N is a decision network for L if and only if:

1. *for every $w \in \Sigma^{\leq n}$, if w is presented to I, then the network reaches an equilibrium with $a_{out} = Yes$ whenever $w \in L$, otherwise it reaches equilibrium with $a_{out} = No$ for some constants, Yes and No.*

2. *If L is recursive then there are no other equilibria.*

So a decision network for a particular finite language is a connectionist network where the inputs are presented with words over an alphabet of maximum and minimum activation values. Note that this small alphabet does not restrict the types of languages that can be accepted by decision networks because any other alphabet can be represented using just two symbols and binary encodings.

For every input, the network's single output unit assumes a value that can be interpreted as either *Yes* or *No*. If the word is in the decision network's language, then the decision network must answer *Yes*. Conversely, if the word is not in the language, and the language is recursive, then the network must answer *No*. If the language is recursively enumerable, but not recursive, then the network needn't answer for otherwise a Turing machine would not be able to simulate the network, leading to a contradiction.

Some connectionists might object to the use of a single output unit, preferring to say instead that the network assumes an accepting pattern on its output units. However, deciding if the output units' pattern is an accepting pattern is really just another language recognition problem; if connectionists want to claim that their networks can accept formal languages, then they should also be able to accept the language of accepting output patterns. Thus, one unit should be sufficient to indicate acceptance of the input.

As noted above, a particular implementation of a Turing machine can accept only a finite language. However, in principle, a Turing machine can be constructed that accepts arbitrarily large subsets of an infinite language. A computational model, such as decision networks, which restricts the size of its input is therefore inferior to the Turing machine model.

For decision networks, one possible solution to the difficulty of finite networks accepting infinite languages is to present symbols to the network's input units sequentially. This solution was proposed by Elman for parsing natural languages (Elman 1990) and it does allow the network to accept words of arbitrary length. However, notice that since the connectionist network has only a finite number of possible states, and is thus essentially a finite automaton, it can accept at most a regular language. That is, while strings of arbitrary length can be input into this type of network, this solution restricts the class of languages that can be accepted by such decision networks to the regular languages.

Admittedly, any realization of a Turing Machine can also only accept a finite, and thus regular language. However, it is easy to create a Turing Machine that can accept a word of arbitrary length simply by providing a longer tape. It is not so easy to extend networks of the type proposed by Elman: no mechanism has been proposed by which such networks can be systematically enlarged.

If, in order to accept large non-regular languages, we must expand a network of the type proposed by Elman, then it is reasonable to limit the length of the input to those words that can be accepted by the network. In other words, decision networks are sufficient, and there is no need to present symbols sequentially to the network's input units. Therefore we can consider, as a solution to accepting strings of arbitrary length, an infinite, uniform family of finite decision networks. Each member of the family accepts a finite subset of the language. For example, the ith decision network would accept or reject words that contain at most i symbols. To decide if a word is in the language, first find an appropriate member of the family, and present the symbols in the word simultaneously to the input units of the network. Such a family of decision networks each deciding a finite subset of the language is called a *decision family* for the language.

Definition 2 *Let L be a language, and let $L_i = \{w | W \in L, |w| \leq i\}$ be the subset of the language consisting of words that have length at most i. Suppose there is an algorithm, which when given i computes N_i, a decision network for L_i. Then the set $\{N_1, N_2, ...\}$ is a decision family for the language L.*

The family must be *uniform* in the sense that given an index i, a Turing machine could write down a description of the ith network in the family. This uniformity is important for two reasons. First, there must be a way to synthesize the decision networks, or to "find an appropriate member of the family." Second, non-uniformity would allow the creation of a family of decision networks that decides a non-recursive language — a language that can't be decided by a Turing machine.

5.1.4 Harmonic Decision Networks

While they are a reasonable model for connectionist systems that accept formal languages, the decision networks described in the previous section do not capture many of the principles of harmony theory. The crucial aspect of harmony networks that distinguishes them from other connectionist models is the energy function that describes the way the harmony network relaxes into certain stable equilibria, and which is used in the design of the harmony network. We can thus define a *harmonic decision network* which combines the principle of energy minimization with decision networks.

Definition 3 *A harmonic decision network is a decision network that admits* $E(a) = -\frac{1}{2}a^T W a$ *as an energy or Liapunov function, where* a *is the activation vector and* W *is the weight matrix.*

The construction of Chapter 4 might suggest that the algorithm that constructs a family of decision networks must use the energy function in some way; however, such a stipulation would be difficult to formalize. The development and use of tensor representations for parse trees might imply a refinement also — that the hidden units should assume a pattern that is systematically related to the input. For example, the definition could assert that the hidden units should find stable equilibria that represent a parse tree for the word presented to the inputs. Neither of these refinements is required, however, by the discussion in the next section about the restrictions on languages that are accepted by a small subclass of harmonic decision networks.

5.1.5 Symmetric Harmonic Decision Networks

This section discusses a special kind of harmonic decision network where the state space is symmetric about the origin, that is, $a \in D$ if and only if $-a \in D$. Note that this type of symmetry is different from the weight matrix symmetry of Section 4.2.1. Harmonic decision networks that have a symmetric state space will be called *symmetric harmonic decision networks.*

For all harmonic decision networks, the energy function is even. That is,

$$\begin{aligned} E(-a) &= -\frac{1}{2}(-a)^T W(-a) \\ &= -\frac{1}{2}(-1)^2 a^T W a \\ &= -\frac{1}{2} a^T W a \\ &= E(a) \end{aligned}$$

The result is that, due to the following lemma, symmetric harmonic decision networks have stable points at both a and $-a$.

Lemma 3 *Let $E(a)$ be an even real-valued function of $\mathbf{R}^n$, and let R be a region such that $a \in R \Rightarrow -a \in R$. Then a is a local minimum of $E(a)$ in R if and only if $-a$ is also a local minimum value of $E(a)$ in R.*

From a decision network point of view, this even energy function has implications on the meaning of the stable equilibria. At equilibrium, the value, a_{out}, of the decision network's output node, can only be either Yes or No. Otherwise, if a_{out} can have more than two values, then there are some ambiguous inputs. Lemma 3 shows that if a_{out} is an equilibrium value of the output node, then so is $-a_{out}$. Thus in a symmetric BSB decision network, $Yes = -No$. This fact is captured more formally by the following lemma:

Lemma 4 *If*

1. *$H = (I \cup J \cup \{out\}, E)$ is a symmetric harmonic decision network for a language, L.*

2. $L \neq \emptyset$ *and* $L \neq \Sigma^*$.

Then

$$Yes = -No$$

Proof: *Because* $L \neq \emptyset$, H *admits a stable equilibrium,* a, *with* $a_{out} = Yes$. *By Lemma 3,* $-a$ *is also an equilibrium point. Because* H *is a harmonic decision network, either* $-a_{out} = Yes$ *or* $-a_{out} = No$. *So either* $Yes = -No$ *or* $Yes = -Yes = 0$.

Similarly, because $L \neq \Sigma^*$, *either* $No = -Yes$ *or* $No = -No = 0$. *In all cases* $Yes = -No$. □

Because the energy is even, $a_{out} = Yes$ in half the energy function's local minima and in the other half, $a_{out} = -Yes = No$. In other words, exactly half the stable equilibria correspond to the network accepting the input word, and half of the minima correspond to rejecting the word. This result is formally presented by Lemma 5.

Lemma 5 *If*

1. $H = (I \cup J \cup \{out\}, E)$ *is a symmetric harmonic decision network with output unit out.*

2. $Yes \neq No$

3. A *is the set of accepting equilibrium states,*

$$A = \{a | a \text{ is an equilibrium state and } a_{out} = Yes\}$$

4. R *is the set of rejecting equilibrium states,*

$$R = \{a | a \text{ is an equilibrium state and } a_{out} = No\}$$

Then

$$|A| = |R|$$

Proof: *Because* $Yes = -No$, *then for every* $a \in A$, $-a \in R$, *so* $|A| \leq |R|$. *Similarly, for every* $a \in R$, $-a \in A$, *so* $|R| \leq |A|$. □

It is important to realize that the presentation of two different words to a decision network may cause the network to reach the same equilibrium point. In particular, the energy function could have only two local minima — one where $a_{out} = Yes$ and another where $a_{out} = No$. On the other hand, some words could correspond to two or more local minima.

However, if each word presented to the input units causes a symmetric harmonic decision network to find a unique local minimum, there would be as many local minima as words over the alphabet. In fact, if every word should correspond to exactly one stable equilibrium of a symmetric harmonic decision network, then it is a straightforward result of Lemma 5 that the language is equal in size to its complement:

Theorem 5 *If*

1. $H = (I \cup J \cup \{out\}, E)$ *is a harmonic decision network for a language* L.
2. H*'s state space is symmetric about the origin. So if* a *is an activation vector of* H*, then* $a \in D^{|I|} \times D^{|J|} \times D$ *where* $D = [-k, k]$.
3. $E \subseteq D^{|I|} \times D^{|J|} \times D$ *is the set of stable equilibria of* H.
4. *There is a bijection* $\varphi : E \rightarrow L \cup \overline{L}$ *between* H*'s equilibrium points and the set of all strings.*

Then

$$|L| = |\overline{L}|$$

Proof: *Let* $A = \{(x, y, Yes) | x \in D^{|I|}, y \in D^{|J|}\}$ *be the set of all accepting stable equilibria, and let* $R = \{(x, y, No) | x \in D^{|I|}, y \in D^{|J|}\}$ *be the set of all rejecting stable equilibria. By Lemma 5,* $|A| = |R|$*. Since* φ *is a bijection, then*

$$|L| = |\varphi(A)| = |\varphi(R)| = |\overline{L}|$$

□

So we see that, for almost all languages, there is no one-to-one correspondence from the set of words onto the set of equilibria. Either some words must correspond

to more than one stable equilibrium in a symmetric harmonic decision network, or several words must map to the same stable equilibrium. If there is no such bijection, it remains to be seen if there are similar restrictions on the size of the language.

Proving such a restriction requires a result relating the evenness of the energy function to oddness of the function calculated by the network. In particular, we need to show that if

1. $H = (V, E)$ is a symmetric harmonic decision network.

2. $h : D^{|V|} \rightarrow D^{|V|}$ is the function calculated by H.

then

$$h(-x) = -h(x)$$

Unfortunately, this hypothesis does not (seem to) follow from the definition of an energy function. We can, however, prove that if H is implemented by one of the types of networks in which we are interested, then it has the desired property.

Recall that the particular models that have the energy function required by harmonic decision networks are special cases of the Brain-State-in-a-Box (BSB) model and the Hopfield model. Both of these models were introduced in Section 2.6, and discussed in more detail in Section 4.2.1.

The BSB model is defined by the equations

$$\begin{aligned} b(t) &= a(t) + \gamma W a(t) \\ a(t+1) &= S(b(t)) \end{aligned}$$

If a harmonic decision network is implemented by a BSB where the threshold function $S()$ is odd so that $S(-a) = -S(a)$, then it is a symmetric harmonic decision network. This property of the threshold function produces the effect that the function computed by the network is symmetric about the origin.

Lemma 6 *If*

1. *H is a BSB network with an odd threshold function, $S(-a) = -S(a)$.*

2. *$a(t)$ is the value of H's activation vector at time t.*

3. *$a'(t) = -a(t)$.*

Then

$$a'(t+1) = -a(t+1)$$

Proof:

$$\begin{aligned} a'(t+1) &= S(a'(t) + \gamma W a'(t)) \\ &= S(-a(t) - \gamma W a(t)) \\ &= S(-(a(t) + \gamma W a(t))) \\ &= -S(a(t) + \gamma W a(t)) \\ &= -a(t+1) \end{aligned}$$

□

Corollary 1 *If*

1. *$H = (V, E)$ is a BSB network, with an odd threshold function, $S(-a) = -S(a)$.*

2. *$h : D^{|V|} \rightarrow D^{|V|}$ is the function calculated by H.*

Then

$$h(-x) = -h(x)$$

Proof: *Let $x'(t) = -x(t)$, by induction $x'(t+k) = -x(t+k)$ for all k. Thus*

$$\begin{aligned} h(-x) &= \lim_{k \to \infty} x'(t+k) \\ &= -\lim_{k \to \infty} x(t+k) \\ &= -h(x) \end{aligned}$$

□

Like the BSB model, the Hopfield model also displays symmetry whenever the activation function is odd. Recall the Hopfield model as described in Section 2.6.

$$\begin{aligned} C_i \frac{du_i}{dt} &= \sum_j W_{i,j} a_j - \frac{u_i}{R_i} + I_i \\ u_i &= g_i^{-1}(a_i) \end{aligned}$$

As revealed in Section 4.2.1, if the activation functions, g_i are steep, and the steady inputs, I_i are all zero, then the Hopfield model has the required energy function. If g_i is an odd function so that $g_i(-x) = -g_i(x)$, then this model displays symmetry similar to the BSB model.

Lemma 7 *If*

1. *H is a Hopfield network with odd activation functions, so that for every i, $g_i(-x) = -g_i(x)$.*
2. *$a(t)$ is the value of H's activation vector at time t.*
3. *let $a(t') = -a(t)$.*

Then

$$\frac{da_i(t')}{dt} = -\frac{da_i(t)}{dt}$$

Proof: *From the defining equations we see that*

$$\begin{aligned} C_i \frac{du_i(t)}{dt} &= \sum_j W_{i,j} a_j(t) - \frac{u_i(t)}{R_i} \\ C_i \frac{dg_i^{-1}(a_i)}{da_i} \frac{da_i(t)}{dt} &= \left(\sum_j W_{i,j} a_j(t) - \frac{u_i(t)}{R_i} \right) \\ \frac{da_i(t)}{dt} &= \frac{1}{C_i \frac{dg_i^{-1}(a_i(t))}{da_i}} \left(\sum_j W_{i,j} a_j(t) - \frac{g_i^{-1}(a_i(t))}{R_i} \right) \end{aligned}$$

So therefore,

$$\frac{da_i(t')}{dt} = \frac{1}{C_i \frac{dg_i^{-1}(a_i(t'))}{da_i}} \left(\sum_j W_{i,j} a_j(t') - \frac{g_i^{-1}(a_i(t'))}{R_i} \right)$$

$$
\begin{aligned}
&= \frac{1}{C_i \frac{dg_i^{-1}(a_i(t'))}{da_i}} \left(-\sum_j W_{i,j} a_j(t) - \frac{-g_i^{-1}(a_i(t))}{R_i} \right) \\
&= \frac{-1}{C_i \frac{dg_i^{-1}(a_i(t'))}{da_i}} \left(\sum_j W_{i,j} a_j(t) - \frac{g_i^{-1}(a_i(t))}{R_i} \right) \\
&= \frac{-1}{C_i \frac{dg_i^{-1}(a_i(t))}{da_i}} \left(\sum_j W_{i,j} a_j(t) - \frac{g_i^{-1}(a_i(t))}{R_i} \right) \\
&= -\frac{da_i(t)}{dt}
\end{aligned}
$$

□

Corollary 2 *If*

1. *H is a Hopfield network with odd activation functions.*

2. *$h : D^{|V|} \rightarrow D^{|V|}$ is the function calculated by H.*

Then

$$h(-x) = -h(x)$$

Proof: *Since $\frac{dx(t)}{dt}$ is odd, then $h(x) = \lim_{t \rightarrow \infty} x(t)$ is also odd.* □

Because symmetric harmonic decision networks are implemented by symmetric BSB or Hopfield networks, which implement odd functions, then symmetric harmonic decision networks will also implement odd functions. This is the result that we require to find the restrictions on the languages that can be accepted by a symmetric harmonic decision network. The following theorem reveals that even if there is no one-to-one correspondence from the words onto the equilibria, the language is still equal in size to its complement.

Theorem 6 *If*

1. *$H = (I \cup J \cup \{out\}, E)$ is a harmonic decision network for a language, L.*

2. *H is either a BSB network or a Hopfield network as in Lemmas 6 and 7.*

Then

$$|L| = |\overline{L}|$$

Proof: *Let* $h : D^{|I|} \times D^{|J|} \times D \rightarrow D^{|I|} \times D^{|J|} \times D$ *be the function calculated by* H.

For $w \in \{-k, k\}^*$, *let* $F_w = \{(x_1, x_2, x_3) | \exists\, y_1, y_2 \; s.t. \; h(w, y_1, y_2) = (x_1, x_2, x_3)\}$ *be the set of all final states for* w.

Because $h(-(x, y, z)) = -h((x, y, z))$ *(due to corollaries 1 and 2),* $F_{-w} = -F_w$.

Thus, for every $u \in L$ *there is a* $v = -u$ *such that* $-F_u = F_v$. *Because* $Yes = -No$ *(lemma 4), then* $v \in \overline{L}$. *So,* $|L| \leq |\overline{L}|$.

Similarly, for every $u \in \overline{L}$ *there is a* $v = -u$ *such that* $-F_u = F_v$, *and so* $v \in L$. *Thus* $|\overline{L}| \leq |L|$. □

Thus, for almost every language, there is no symmetric harmonic decision network that decides that language.

5.2 Network Size

The previous section investigated the difficulties of providing input to, and determining output from harmony networks. A second problem with harmony networks involves the number of units required by the tensor representation held by the network. This number, we will see, is quite large.

Section 3.3 showed that the role vectors in the tensor representation for the parse tree are constructed recursively. If the base role vectors are of dimension r, a symbol in the ith level of the tree will appear in a role of dimension r^i. Two vertices at different levels in the tree will appear in roles with different dimensions; these are added by taking their direct sum. The result is that the dimension of the sum of the two representations will be the sum of their dimensions.

The total number of units required to represent a parse tree of depth k is then

$$\begin{aligned} N(k) &= \sum_{i=0}^{k-1} f r^i \\ &= f\frac{1 - r^k}{1 - r} \end{aligned}$$

Here, f is the dimension of the symbols. Thus, if the roles and symbols are both two-dimensional, then the number of units required to represent a tree of depth 30 is over 2×10^{10}. Because the grammar has a special form where each production has at most two symbols on the right (see Section 4.1), parse trees of relatively short strings in formal languages will be much deeper than thirty levels. Moreover, if English sentences are to be parsed using a grammar that allows at most two symbols on the right of each production, then complicated sentences, such as this one, will likely require a parse tree with more than thirty levels even if words are taken to be atomic and have no so-called micro-features as proposed by Smolensky (1988).

Harmony networks were never meant as a metaphor for the human brain, although the proponents claim that the level of harmony networks is much "closer to the neural level" than symbolic algorithms (Legendre, Miyata, and Smolensky 1990). Consider though that the number of neurons in the brain is estimated to be between 10^{10} and 10^{11} (Rumelhart and McClelland 1986). So, even with minimal assumptions about the dimensions of the symbols and the roles, the number of units required to represent relatively small parse trees approaches the number of neurons in the brain.

Intuitively, the number of units required is much greater than we would expect. One reason for this is that the representation is capable of capturing complete binary trees — binary trees where every non-leaf vertex has both a left- and a right-child. A quick examination of the parse tree in Figure 4.2 will reveal that this representational capacity is not required. In the parse trees resulting from the grammars discussed in Section 3.4.1, vertices typically have only a left-child, while a few have a right-child as well. The result is that much of the representational capacity of the tensor representation is wasted.

There are a number of natural representations that could be proposed to reduce this waste. For example, each level of the parse tree could be represented by a group of units, as it is in the tensor representation, which economically encodes the symbols on that level. While it might work, it is difficult to see how such a network could find maximum-harmony trees without resorting to directly manipulating symbols, and implementing classical theories — a pitfall that harmony theorists want to avoid if they are to successfully answer Fodor and Pylyshyn's challenge.

Another solution to the size problem is to propose a representation that does not grow exponentially with the depth of the recursion. For example, Plate's Holographic Representations (Plate 1993) could offer a reprieve to harmony theory. However, the next section will show that, in their present form, harmony networks admit stable equilibria that do not represent parse trees, and so, the question of size is academic.

5.3 Harmony Networks Do Not Work

In the previous two sections, we discussed the problems related to connecting a harmony network to the outside world, and the large size of harmony networks. Even if we are willing, however, to ignore these stumbling blocks and carry on to implement a harmony network, we will find that it does not actually perform as advertised: Harmony networks find stable equilibria that do not represent maximum harmony parse trees.

Recapitulating the developments of chapters 3 and 4 will reveal the reason why Harmony networks do not actually find maximum harmony parse trees. The previous two chapters demonstrate

1. A method to represent parse trees using the activation vector of a connectionist network (Section 3.3).

2. A harmony function defined on parse trees. A parse tree is valid if its harmony value is zero. The harmony values of all invalid parse trees are below zero. In other words, the parse tree is valid if and only if its harmony value is a maximum (Section 4.1.1).

3. An energy function, defined on the values of the activation vector. Two models of connectionism — the Brain-State-in-a-Box (BSB) model and the Hopfield model — seek local minima in this energy function (Section 4.2.1).

4. A system for determining the weight matrix of a connectionist network so that if its activation vector is interpreted as a parse tree, then its energy function is the negation of the harmony of that parse tree (sections 4.2.2 and 4.2.3).

The problem is that the energy function is defined on the values of the activation vector. By contrast, the harmony function is defined on possible parse trees. As shown in Section 5.3.1 these two domains are not equal; there are some activation vectors that do not represent any parse tree.

So, while it may be the case that the energy and harmony functions are negations of one another, it is not likely the case that a local minimum of one is a local maximum of the other. Indeed, item 4 above merely guarantees that the energy function passes through zero at the appropriate points, but its minima are unrestricted. More succinctly, the harmony network will find minima that are not even parse trees, let alone valid parse trees.

The reason why harmony networks do not work is straightforward. Section 5.3.2 shows that the weight matrix must have only negative eigenvalues, for otherwise the network constructs structures which are not valid trees. Section 5.3.3 shows that if the weight matrix has only negative eigenvalues, then the energy function admits only a single zero — the origin. Furthermore, we show that the origin cannot be interpreted as a valid parse tree. Thus, the stable equilibria of a harmony network are not all valid parse trees.

5.3.1 More Activations Than Trees

As noted above, the reason why harmony networks do not work is that they seek minima in their state space which may not coincide with parse tree representations. One way to ameliorate this would be by making every possible activation vector represent some parse tree. In other words, the parse tree representations fill the state space of the harmony network with the same density as the activation vectors. If every activation vector represents some parse tree, then the rules that determine the weight matrix — and hence determine the energy function — will ensure that the absolute energy minima agree with the valid parse trees.

Unfortunately, if every activation vector represents some parse tree, then the system of equations in table 4.3 has no solution. The first four lines of that table indicate

that for each terminal or non-terminal, f, W_{root} must satisfy the equation:

$$(f + \vec{0} \oslash r_l)^T W_{root}(f + \vec{0} \otimes r_l) = h$$

where $h \in \{2,4,6\}$. If the symbols of the grammar are two dimensional, then there will be symbols represented by each vector, $(x_1,x_1),(x_1,x_2),(x_2,x_1)$, and (x_2,x_2), where $x_1 \neq x_2$. Therefore, W_{root} must satisfy the equations,

$$\left.\begin{aligned} x_1^2(W_{root_{11}} + W_{root_{12}} + W_{root_{21}} + W_{root_{22}}) &= h_1 \\ x_1^2 W_{root_{11}} + x_1x_2 W_{root_{12}} + x_1x_2 W_{root_{21}} + x_2^2 W_{root_{22}} &= h_2 \\ x_2^2 W_{root_{11}} + x_1x_2 W_{root_{12}} + x_1x_2 W_{root_{21}} + x_1^2 W_{root_{22}} &= h_3 \\ x_2^2(W_{root_{11}} + W_{root_{12}} + W_{root_{21}} + W_{root_{22}}) &= h_4 \end{aligned}\right\} \text{where } h_i \in \{2,4,6\}$$

Because $h_i \in \{2,4,6\}$, there must be a pair h_i, h_j which are equal. In that case, it can be shown using Gaussian elimination, there is no solution for $W_{root_{11}}, W_{root_{12}}, W_{root_{21}}, W_{root_{22}}$. If the symbols are represented by vectors of dimension three or greater, then the same contradiction occurs. The result is that the weight matrix cannot be derived.

Thus there are some activation vectors that do not represent any tree — valid or invalid. The question now becomes one of determining whether all of the harmony network's stable equilibria are valid parse trees.

5.3.2 Non-Negative Eigenvalues Yield Non-Trees

The last section showed that not every point in the harmony network's state space can represent a parse tree. However, the harmony network can be restricted so that all its stable equilibria fall on the corners of its state space. With such an affinity for the corners, the interior points need not represent parse trees. While the previous section showed that not every corner can represent a parse tree (simply let x_1, x_2 be the minimum and maximum activation values), not every corner will be a stable equilibrium, and so, such a restriction could force the harmony networks to find maximum harmony parse trees.

Golden (1986) showed that making all the weight matrix's eigenvalues non-negative will ensure that the BSB model's stable points lie in the corners. Hopfield (1984)

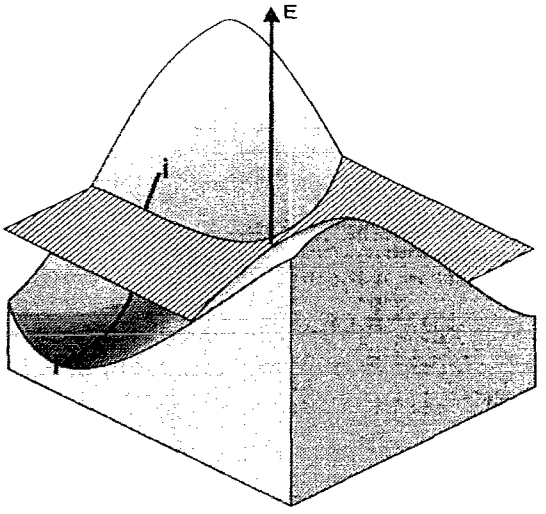

Figure 5.1: The energy function of a two-dimensional harmony network where one eigenvalue is negative and the other positive. The hashed plane represents the plane $E = 0$. It intersects the energy function and the vertical axis at the origin. The points i and f respectively represent an initial and a final state of the network.

claimed that "usually" the Hopfield model's stable points lie in the corners. Experiments show, however, that when the weight matrix admits negative eigenvectors, the stable points tend to be away from the corners. On the other hand, non-negative eigenvalues do tend to force the Hopfield model to find only corners.

If any of the eigenvalues of the weight matrix, W, is positive, then it is easy to show that the harmony network will seek a stable equilibrium that does not represent a parse tree at all. Let $\lambda > 0$ be a positive eigenvalue of W, and let e be an eigenvector, corresponding to λ, that falls within the state space. Then,

$$\begin{aligned} E(e) &= -\frac{1}{2}e^T We \\ &= -\frac{1}{2}e^T \lambda e \\ &= -\frac{1}{2}\lambda e^T e \\ &\leq 0 \end{aligned}$$

Figure 5.1 illustrates the energy function of a harmony network where one eigenvalue is positive. In this figure, all the valid parse trees would rest on the hashed zero-energy

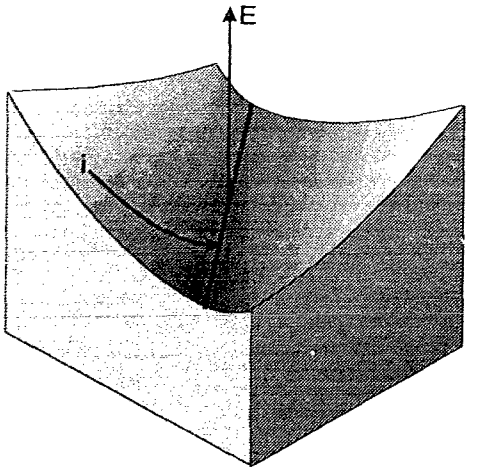

Figure 5.2: The energy function of a two-dimensional harmony network where one eigenvalue is negative and the other is zero. The heavy line represents the intersection of the surface with the plane $E = 0$. It intersects the vertical axis at the origin. The points i and f respectively represent an initial and a final state of the network.

plane, and all the invalid trees would be above it. Because the energy function drops below zero, the harmony network would have to undergo an energy increase in order to find a zero-energy stable equilibrium. This cannot happen, and so, the network reaches an equilibrium with energy strictly less than zero. Therefore the harmony network with positive eigenvalues will certainly find stable equilibria which are not valid parse tree representations.

Now, suppose W, the weight matrix, has a zero eigenvalue. If e is an eigenvector corresponding to that eigenvalue, then for every real α, $\alpha We = 0$. Then one of the following must be true:

αe is not a stable equilibrium: In that case, the energy function must drop below zero, yielding a sub-zero stable equilibrium — a stable equilibrium that does not represent any tree.

αe is a stable equilibrium: Then for every α, αe must be a valid tree representation. This situation is represented in Figure 5.2, where the set of all points, αe, is represented by the heavy line. This implies that there is a symbol, $(a_1, a_2, \ldots, a_n)$,

such that $\alpha_1(a_1,\ldots,a_n)$, $\alpha_2(a_1,\ldots,a_n),\ldots,\alpha_{n^2+1}(a_1,\ldots,a_n)$ are also all symbols. As before, this implies that W_{root} must satisfy the equations,

$$\begin{aligned}((a_1,\ldots,a_n)+\vec{0}\otimes r_l)^T W_{root}((a_1,\ldots,a_n)+\vec{0}\otimes r_l) &= \frac{h_1}{\alpha_1^2}\\ ((a,\ldots,a_n)+\vec{0}\otimes r_l)^T W_{root}((a,\ldots,a_n)+\vec{0}\otimes r_l) &= \frac{h_2}{\alpha_2^2}\\ &\vdots\\ ((a,\ldots,a_n)+\vec{0}\otimes r_l)^T W_{root}((a,\ldots,a_n)+\vec{0}\otimes r_l) &= \frac{h_{n^2+1}}{\alpha_{n^2+1}}\end{aligned}$$

where $h_i \in \{2,4,6\}$. By Gaussian elimination, there is no solution to this system of equations, and hence, there is no such weight matrix.

In all cases, if the weight matrix has non-negative eigenvalues, then the harmony network admits stable equilibria that do not represent any tree. Thus, the eigenvalues must all be negative.

5.3.3 Negative Eigenvalues Yields Non-Tree

If all the eigenvalues of the weight matrix are negative, then the energy function has a very special shape: it is a paraboloid centered on the origin and concave in the direction of positive energy. This is easily seen by considering the first and second derivatives of E:

$$\begin{aligned}\frac{\partial E(\vec{x})}{\partial x_i} &= \frac{\partial}{\partial x_i}\left(\sum_j\sum_k -\frac{1}{2}W_{j,k}x_j x_k\right)\\ &= \frac{\partial}{\partial x_i}\left(-\frac{1}{2}\sum_j\sum_{k>j}2W_{j,k}x_j x_k - \frac{1}{2}\sum_j W_{j,j}x_j^2\right)\\ &= -\sum_j W_{i,j}x_j\\ \frac{\partial^2 E(\vec{x})}{\partial x_i \partial x_j} &= \frac{\partial}{\partial x_j}\left(-\sum_k W_{i,k}x_k\right)\\ &= -W_{i,j}\end{aligned}$$

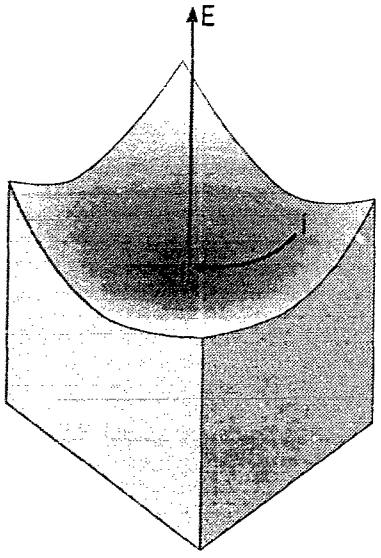

Figure 5.3: The energy function of a two-dimensional harmony network where both eigenvalues are negative. The vertical axis pierces the surface at the origin. The points i and f respectively represent an initial and a final state of the network.

Clearly, all the first derivatives are zero at the origin, and so, it is a critical point. Now the origin is a strict minimum if all the roots of the following equation are positive (see for example Taylor and Mann 1983, p218):

$$\begin{aligned} 0 &= \det \begin{vmatrix} \frac{\partial^2 E(\vec{x})}{\partial x_1 \partial x_1} - \lambda & \frac{\partial^2 E(\vec{x})}{\partial x_1 \partial x_2} & \cdots \\ \frac{\partial^2 E(vecx)}{\partial x_2 \partial x_1} & \frac{\partial^2 E(vecx)}{\partial x_1 \partial x_2} - \lambda & \\ \vdots & & \ddots \end{vmatrix} \\ &= \det \begin{vmatrix} -W_{1,1} - \lambda & -W_{1,2} & \cdots \\ -W_{2,1} & -W_{2,2} - \lambda & \\ \vdots & & \ddots \end{vmatrix} \\ &= \det |-W - \lambda I| \end{aligned}$$

This last equation is known as the characteristic polynomial of $-W$, and its roots are the eigenvalues of $-W$. Therefore, if λ is a root then it is also an eigenvalue of $-W$, or equivalently, it is the negative of an eigenvalue of W. Because all of W's eigenvalues are negative, the origin is a strict minimum, and indeed it is the only minimum. Such a harmony network is represented by Figure 5.3.

As can be seen, the origin is the only stable point where the energy is zero. But it cannot represent a parse tree which is valid for the grammar, for then,

$$S + T_L \otimes r_l + T_R \otimes r_r = (0, \ldots, 0)$$

where T_L, T_R are appropriate left and right subtree representations. Because each of the subtrees is multiplied by either r_l or r_r, they are not the same dimension as S, and are consequently concatenated instead of added. Therefore $S = \vec{0}$. But then, W_{root} must satisfy the equation

$$(\vec{0} + \vec{0} \otimes r_l) W_{root} (\vec{0} + \vec{0} \otimes r_l) = -2$$

This is impossible, and so, the origin is not a valid tree representation.

5.3.4 Conclusion

This section has shown that in every case, a harmony network will reach stable equilibria that are not valid parse trees. This is not unexpected. Because the energy function is a very simple function, it would be more surprising if such a connectionist system could construct complicated structures such as parse trees for a context free grammar.

Chapter 6

Resolution

The previous section showed that harmony networks do not construct only maximum-harmony parse trees for some context free grammar. Even so, this fault does not spell the end of harmony theory, but only upsets a particular implementation of it. The basic idea — that of describing the operation of a connectionist system at a high level, and then using that high level description to forge a connection with an even higher symbolic level — still seems more likely to yield neurally plausible explanations for high level cognitive functions than the ad hoc approaches proposed by many connectionists (for example, Shastri and Ajjanagadde 1993). However, the particular type of connectionist system postulated by harmony theory is too simple to uphold the bold claims of its proponents.

It is possible that some future harmony network will be constructed that takes advantage of a more complicated connectionist system (perhaps one where the weight matrix is not symmetric) and the resulting energy function to create a harmony network that actually does construct only valid parse trees. Such a network would still only answer one of the main difficulties exposed by this thesis.

A successful implementation of a harmony network will still be extremely large if it is to construct non-trivial parse trees. The network's magnitude may find a solution in one of the other techniques for convolving symbols with their roles. For example, while they do not allow perfect decomposition, Plate's holographic representations do not grow exponentially with the depth of the recursion.

Moreover, even a successful harmony network will still require input in the form of parse trees, and will not offer any meaningful output. The harmony network cannot be said to accept a formal language because its inputs are parse trees, and the output does not directly indicate acceptance or rejection. The question of output may be especially difficult for connectionists; particularly in less constrained formal languages, such as recursive languages, membership is a global property of the entire string, not of local portions. Connectionist models, by contrast, perform only local operations, and so, it is difficult to see how they can capture the full generality of formal languages. Harmony theorists will have to answer this objection before claiming to have uncovered a new means by which connectionist networks can accept formal languages.

Perhaps a more interesting question is whether or not harmony networks in their present form successfully refute Fodor and Pylyshyn's challenge (Fodor and Pylyshyn 1988). That is, do harmony networks account for the systematicity and productivity of human cognition without directly implementing symbolic algorithms?

At first, it seems that they do. Provided a network actually did find stable equilibria that represented valid parse trees, we would have to concede that the network was exhibiting some amount of systematicity and productivity. That is, if it is capable of reaching a particular stable equilibrium, then a harmony network must also be able to reach a host of other systematically related equilibria. This property is guaranteed by the recursive construction of the harmony network, which allows a legal subtree to appear in any legal position of the parse tree represented by the stable equilibrium. What's more, provided the network is big enough to represent them, it will be able to represent every possible valid parse tree as a stable equilibrium. Thus a successful harmony network would be able to entertain an infinite number of systematically related thoughts, satisfying Fodor and Pylyshyn's requirement for productivity.

So, it might appear that the harmony networks do satisfy Fodor and Pylyshyn's original challenge. This is misleading. The fact that they do not actually have an input and create meaningful output from it, however, means that harmony networks do not actually calculate anything. In other words, while they may be able to token an infinite number of systematically related thoughts, harmony networks cannot produce

any inferences, and so, they do not account for a property which Fodor and Pylyshyn call the "systematicity of inference." This quality of human cognition relates to the idea that someone who can draw one kind of inference can necessarily compute a number of other similar inferences. Because harmony networks do not calculate any inference, they satisfy this property only trivially, and so, harmony networks do not exhibit the spirit of systematicity of inference.

Even so, we should not abandon the principles that motivate harmony networks. Considering connectionist models as energy-minimization systems could still lead to a successful answer to the issues raised by Fodor and Pylyshyn. As in the harmonic decision networks that were introduced in this thesis, however, the whole network — including input and output units — should be included when considering energy reduction.

Bibliography

Anderson, J., J. Silverstein, S. Ritz, and R. Jones (1977, Sept.). Distinctive features, categorical perception, and probability learning: Some applications of a neural model. *Psychological Review 84*(5), 413–451.

Cohen, M. and S. Grossberg (1983, Sept.). Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Transactions on Systems, Man and Cybernetics SMC-13*(5), 815–826.

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

Fisher, R. C. (1970). *An Introduction to Linear Algebra.* Encino, California: Dickenson Pub.

Fodor, J. and B. McLaughlin (1990). Connectionism and the problem of systematicity: Why Smolensky's solution doesn't work. *Cognition 35*, 183–204.

Fodor, J. and Z. Pylyshyn (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition 35*, 183–204.

Golden, R. (1986). The 'brain-state-in-a-box' neural model is a gradient descent algorithm. *Journal of Mathematical Psychology 30*, 73–80.

Golden, R. (1988). A unified framework for connectionist systems. *Biological Cybernetics 59*, 109–120.

Hinton, G., J. McClelland, and D. Rumelhart (1986). Distributed representations. In D. Rumelhart, J. McClelland, and The PDP Research Group (Eds.), *Parallel*

Distributed Processing Explorations in the Microstructure of Cognition Volume 1: Foundations, pp. 77–109. Cambridge: MIT Press.

Hopcroft, J. and J. Ullman (1979). *Introduction to Automata Theory, Languages and Computation.* Massachusetts: Addison-Wesley.

Hopfield, J. (1982, April). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Science 79*, 2554–2558.

Hopfield, J. (1984, May). Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Science 81*, 3088–3092.

Hopfield, J. (1987, Dec.). Learning algorithms and probability distributions in feed-forward and feed-back networks. *Proceedings of the National Academy of Science 84*, 8429–8433.

Legendre, G., Y. Miyata, and P. Smolensky (1990). Harmonic grammar – a formal multi-level connectionist theory of linguistic well-formedness: Theoretical foundations. In *Proceedings of the Twelfth National Conference on Cognitive Science*, Cambridge, MA, pp. 385–395. Lawrence Erlbaum.

Leipholz, H. (1987). *Stability Theory An Introduction to the Stability of Dynamic Systems and Rigid Bodies* (Second ed.). New York: John Wiley.

Li, J., A. Michel, and W. Porod (1988). Qualitative analysis and synthesis of a class of neural networks. *IEEE Transactions on Circuits and Systems 35*(8), 976–985.

Lillo, W., D. Miller, S. Hui, and S. Zak (1994, Sept.). Synthesis of brain-state-in-a-box (BSB) based associative memories. *IEEE Transactions on Neural Networks 5*(5), 730–737.

Michel, A., J. Farrell, and W. Porod (1989). Qualitative analysis of neural networks. *IEEE Transactions on Circuits and Systems 36*(2), 229–243.

Plate, T. A. (1993). Holographic recurrent networks. In S. Cowan and C. Giles (Eds.), *Advances in Neural Information Processing Systems 5*, pp. 34–42. San Mateo: Morgan Kaufmann.

Proskurowski, A. (1981). Recursive graphs, recursive labellings and shortest paths. *SIAM Journal of Computing 10*(2), 0.

Rumelhart, D., G. Hinton, and J. McClelland (1986). A general framework for parallel distributed processing. In D. Rumelhart, J. McClelland, and The PDP Research Group (Eds.), *Parallel Distributed Processing Explorations in the Microstructure of Cognition Volume 1: Foundations*, pp. 77–109. Cambridge: MIT Press.

Rumelhart, D. and J. McClelland (1986). PDP models and general issues in cognitive science. In D. Rumelhart, J. McClelland, and The PDP Research Group (Eds.), *Parallel Distributed Processing Explorations in the Microstructure of Cognition Volume 1: Foundations*, pp. 110–149. Cambridge: MIT Press.

Salam, F., Y. Wang, and M.-R. Choi (1991, Feb.). On the analysis of dynamic feedback neural nets. *IEEE Transactions on Circuits and Systems 38*(2), 196–201.

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

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. In D. Rumelhart, J. McClelland, and The PDP Research Group (Eds.), *Parallel Distributed Processing Explorations in the Microstructure of Cognition Volume 1: Foundations*, pp. 77–109. Cambridge: MIT Press.

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

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

Smolensky, P. (1991). Connectionism, consitituency and the language of thought. In B. Loewer and G. Rey (Eds.), *Meaning in Mind: Fodor and his Critics*, pp. 201–227. Oxford: Basil Blackwell.

Smolensky, P. (1993). Harmonic grammars for formal languages. In S. Hanson, J. Cowan, and C. Giles (Eds.), *Advances in Neural Information Processing Systems 5*, pp. 847–854. San Mateo: Morgan Kauffman.

Smolensky, P., G. Legendre, and Y. Miyata (1992). Principles for an integrated connectionist/symbolic theory of higher cognition. Technical Report CU-CS-600-92, University of Colorado Computer Science Department.

Smolensky, P., G. Legendre, and Y. Miyata (1994). Integrating connectionist and symbolic computation for the theory of language. In V. Honavar and L. Uhr (Eds.), *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration*, pp. 509–530. Boston: Academic Press.

Taylor, A. and R. Mann (1983). *Advanced Calculus* (Third ed.). New York: John Wiley and Sons.

Tesar, B. and P. Smolensky (1994). Synchronous firing variable binding is tensor product representation with temporal role vectors. In A. Ram and K. Eiselt (Eds.), *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, New Jersey, pp. 870–. Lawrence Erlbaum.

Yang, H. and T. Dillon (1994, Sept.). Exponential stability and oscillation of Hopfield graded response neural network. *IEEE Transactions on Neural Networks 5*(5), 719–729.

Yokonuma, T. (1977). *Tensor Spaces and Exterior Algebra.* Rhode Island: American Mathematical Society.