Distributed Representations and Nested Compositional Structure

by

Tony A. Plate

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

©Copyright by Tony Plate 1994

Distributed Representations and
Nested Compositional Structure

by

Tony A. Plate
Doctor of Philosophy,
Department of Computer Science,
University of Toronto,
1994

Abstract

Distributed representations are attractive for a number of reasons. They offer the possibility of representing concepts in a continuous space, they degrade gracefully with noise, and they can be processed in a parallel network of simple processing elements. However, the problem of representing nested structure in distributed representations has been for some time a prominent concern of both proponents and critics of connectionism [Fodor and Pylyshyn 1988; Smolensky 1990; Hinton 1990]. The lack of connectionist representations for complex structure has held back progress in tackling higher-level cognitive tasks such as language understanding and reasoning.

In this thesis I review connectionist representations and propose a method for the distributed representation of nested structure, which I call "Holographic Reduced Representations" (HRRs). HRRs provide an implementation of Hinton's [1990] "reduced descriptions". HRRs use circular convolution to associate atomic items, which are represented by vectors. Arbitrary variable bindings, short sequences of various lengths, and predicates can be represented in a fixed-width vector. These representations are items in their own right, and can be used in constructing compositional structures. The noisy reconstructions extracted from convolution memories can be cleaned up by using a separate associative memory that has good reconstructive properties.

Circular convolution, which is the basic associative operator for HRRs, can be built into a recurrent neural network. The network can store and produce sequences. I show that neural network learning techniques can be used with circular convolution in order to learn representations for items and sequences.

One of the attractions of connectionist representations of compositional structures is the possibility of computing without decomposing structures. I show that it is possible to use dot-product comparisons of HRRs for nested structures to estimate the analogical similarity of the structures. This demonstrates how the surface form of connectionist representations can reflect underlying structural similarity and alignment.

Acknowledgements

Thanks to Geoff Hinton, my supervisor, for inspiration, guidance, encouragement and support. Geoff always got me going again when I felt discouraged, and inspired me with his curiosity and imagination. Thanks also to Jordan Pollack, for getting me interested in connectionism, and for spending many fascinating hours discussing these ideas with me in New Mexico.

Peter Dayan did an extraordinary job reading several drafts of this thesis and providing insightful comments about everything, ranging from overall organization to individual ideas. Peter's comments improved this thesis immeasurably. Thanks also to Graeme Hirst for a careful reading of the thesis, and many helpful comments. Allan Jepson, Hector Levesque, Ben Murdock, Jeff Siskind, and Paul Smolensky, my other committee members, also gave me much valuable feedback. Thanks also to Arthur Markman and Dedre Gentner for useful and encouraging email discussions about analogy.

Thanks to the Department of Computer Science for its generous support during the PhD program, and thanks also to Canada for the education and for being a great place to live.

The friends I've made at school have made the past six years very enjoyable. Sue Becker, Peter Dayan, Sid Fels, Conrad Galland, Sageev Oore, Chris Williams, Rich Zemel, and the other past and present members of the neuron group have been a great bunch to work with and hang out with.

Thanks to my parents, Dieter and June Plate, for instilling me with a love of learning and always encouraging me in my education.

Thanks to Tina, for support and friendship through some of the most trying moments of this thesis, and to Mary and John and the gang for friendship and fun at the farm.

Contents

Abstract . . . iii
Acknowledgements . . . iv
Contents . . . v
List of Figures . . . xi
List of Tables . . . xii

1 Introduction **1**
1.1 Representations, descriptions, and implementations . . . 2
1.2 Connectionist models . . . 2
1.3 Representational issues . . . 3
1.3.1 Representational adequacy . . . 3
1.3.2 Computational properties . . . 4
1.3.3 Scaling . . . 5
1.3.4 Learning representations for new concepts . . . 6
1.3.5 Other representational issues . . . 6
Sophistication of knowledge representations . . . 6
The Knowledge Representation Hypothesis . . . 6
Logical soundness and completeness vs. computational tractability . 6
Meta-level reasoning . . . 7
Procedural control . . . 7
1.4 Connectionist representations . . . 7
1.4.1 Local representations . . . 7
Advantages of local representations . . . 7
Problems with local representations . . . 8
1.4.2 Distributed representations . . . 9
Advantages of distributed representations . . . 10
Problems with distributed representations . . . 11
1.4.3 Relationships between local and distributed representations . . . 12
1.5 Reduced Descriptions . . . 13
1.6 Discussion . . . 16

2 Review of connectionist and distributed memory models **18**
2.1 Localist connectionist models . . . 18
2.1.1 Interpreting and parsing language by constraint satisfaction . . . 19
Waltz and Pollack: "Massively Parallel Parsing" . . . 19
Selman and Hirst: parallel parsing network . . . 21
Charniak and Santos: parsing network . . . 23
2.1.2 Simple reasoning . . . 23

Shastri: semantic networks . . . 23
Derthick: μKLONE . . . 24
2.1.3 Analogical mapping . . . 25
Holyoak and Thagard: analogical mapping by constraint satisfaction 25
2.2 Distributed connectionist models for simple structure . . . 26
2.2.1 Learning associations between objects . . . 26
Hinton: implementation of semantic networks . . . 26
Anderson, Spoehr and Bennet: Arithmetic . . . 29
Rumelhart and McClelland: learning past-tenses of verbs . . . 29
2.2.2 Learning distributed representations for objects . . . 30
Hinton: "Family trees" network . . . 30
Other networks which learn distributed representations . . . 32
2.2.3 Conjunctive coding . . . 33
Touretzky and Geva: DUCS . . . 36
2.3 Connectionist models which learn to process language . . . 37
2.3.1 Non-recurrent networks . . . 38
Sejnowski and Rosenberg: NETtalk . . . 38
2.3.2 Recurrent networks . . . 39
Elman: Learning to remember . . . 40
St. John and McClelland: sentence comprehension . . . 44
Sopena: parsing sentences with embedded clauses . . . 45
Servan-Schreiber *et al*: Graded State Machines . . . 46
Other recurrent nets for learning regular and context-free languages 47
2.4 Distributed connectionist models for explicit representation of structure . . 47
2.4.1 Touretzky and Hinton: DCPS . . . 48
2.4.2 Pollack: RAAMs . . . 49
2.4.3 Smolensky: Tensor products . . . 52
Dolan and Smolensky: reimplementing DCPS with tensor products 52
Role-filler tensor representations for predicates . . . 52
Dolan: CRAM . . . 54
Halford *et al*: component product representation for predicates . . . 54
2.4.4 Convolution-based models . . . 55
Human memory modelling: models and tasks . . . 57
Liepa: CADAM models . . . 58
Murdock: TODAM . . . 59
Metcalfe: CHARM . . . 60
Murdock: TODAM2 . . . 61

3 Holographic Reduced Representations **63**
3.1 Circular convolution and correlation . . . 63
3.1.1 Distributional constraints on the elements of vectors . . . 65
3.1.2 Why correlation decodes convolution . . . 65
3.1.3 Relationship of convolution to correlation . . . 66
3.1.4 How much information is stored in a convolution trace? . . . 66
3.2 Superposition Memories . . . 67
3.2.1 The probability of correct recognition with superposition memories . 67
3.2.2 The use of thresholds . . . 69
3.3 The need for clean-up memories . . . 69

3.4 Representing complex structure . . . 70
3.4.1 Sequences . . . 70
3.4.2 Chunking of sequences . . . 72
3.4.3 Variable binding . . . 72
3.4.4 Holographic Reduced Representations for simple predicates . . . 72
3.4.5 Holographic Reduced Representations for nested predicates . . . 73
3.4.6 Equivalence of first-order and higher-order representations for roles 74
3.4.7 What can we do with reduced representations? . . . 75
3.5 Constraints on the vectors and the representation of features and tokens . . 75
3.5.1 The representation of features, types, and tokens . . . 75
3.5.2 Why binary distributed representations are difficult to use . . . 76
3.5.3 Normalization of vectors . . . 77
3.5.4 Are the constraints on vectors reasonable? . . . 78
3.6 Mathematical Properties . . . 78
3.6.1 Distributions of dot-products . . . 79
3.6.2 Using FFTs to compute convolution . . . 80
3.6.3 Approximate and exact inverses in the frequency domain . . . 81
3.6.4 Why the exact inverse is not always useful . . . 82
3.6.5 The convolutive power of a vector . . . 83
3.6.6 Matrices corresponding to circular convolution . . . 83
3.6.7 Non-commutative variants and analogs of convolution. . . . 84
3.6.8 Partial derivatives of convolutions . . . 84
3.7 Faster comparison with items in clean-up memory . . . 84
3.8 The capacity of convolution memories and HRRs . . . 86
3.9 Convolution-based memories versus matrix-based memories . . . 87
3.9.1 Comparison of convolution and matrix-based memories . . . 88
3.9.2 Convolution as a low-dimensional projection of a tensor product . . 89
3.10 An example of encoding and decoding HRRs . . . 89
3.10.1 Representation and similarity of tokens . . . 90
3.10.2 Representation and similarity of frames . . . 91
3.10.3 Extracting fillers and roles from frames . . . 91
3.10.4 Probabilities of correct decoding . . . 93
3.10.5 Similarity-preserving properties of convolution . . . 95
3.11 Discussion . . . 97
3.11.1 Learning and HRRs . . . 97
3.11.2 HRRs and Hinton's reduced descriptions . . . 98
3.11.3 Chunking . . . 98
3.11.4 Comparison to other distributed representations for predicates . . . 98
3.12 Conclusion . . . 100

4 HRRs in the frequency domain **101**
4.1 Circular vectors . . . 101
4.2 HRR operations with circular vectors . . . 101
4.3 Comparison with the standard system . . . 103

5 Using convolution-based storage in systems which learn **106**
5.1 Trajectory-association . . . 106
5.1.1 Decoding trajectory-associated sequences . . . 107
5.1.2 Capacity of trajectory-association . . . 108
5.2 Encoding and decoding systems for trajectory-associated sequences . . . 109
5.3 Trajectory-association decoding in a recurrent network . . . 110
5.3.1 Architecture . . . 110
5.3.2 Unit functions . . . 111
5.3.3 Objective function . . . 111
5.3.4 Derivatives . . . 112
5.3.5 Training and testing productive capacity . . . 112
5.4 Simple recurrent networks . . . 113
5.5 Training and productive capacity results . . . 114
5.5.1 HRN results . . . 114
5.5.2 SRN results . . . 116
5.5.3 General training and testing issues . . . 117
5.6 Trajectories in continuous space . . . 117
5.7 Hierarchical HRNs . . . 119
5.8 Discussion . . . 119
5.8.1 Application to processing sequential input . . . 119
5.8.2 Application to classification of sequences . . . 120
5.8.3 Improving the capacity of convolution-based memory . . . 120
5.9 Conclusion . . . 121

6 Estimating analogical similarity **122**
6.1 Models of Analogy Retrieval . . . 122
6.2 An experiment with shape configurations . . . 125
6.2.1 HRRs and dot-product similarity estimates . . . 125
6.3 Analogies between multiple relations . . . 127
6.3.1 Aspects of similarity . . . 128
6.3.2 Classifications of similarity . . . 129
6.3.3 Ratings of the episodes . . . 130
6.4 Estimating similarity by dot-products of HRRs . . . 131
6.4.1 Experiment 1 . . . 131
6.4.2 Conditions for structural similarity to affect the HRR dot-product . . 133
6.4.3 Why the HRR dot-product reflects structural similarity . . . 133
6.4.4 HRRs and multiple occurrences of objects . . . 135
6.4.5 Weighting different aspects of the HRR . . . 135
6.4.6 Covariance of HRR dot-products . . . 136
6.4.7 Properties of the HRR dot-product as a measure of similarity . . . 136
6.5 Contextualized HRRs . . . 137
6.5.1 Flexible salience of object-level isomorphism . . . 138
6.6 Estimating similarity by dot-products of contextualized HRRs . . . 138
6.6.1 Experiments 2 and 3 . . . 138
6.6.2 Discussion of results . . . 138
6.6.3 Extra binding chains introduced by contextualization . . . 139
6.6.4 Limits of contextualization . . . 140
6.7 Interpretations of an analogy . . . 140

6.8 Discussion . . . 142
6.8.1 The criteria for good analogies in SME and ARCS . . . 142
6.8.2 Flexibility of comparisons . . . 144
6.8.3 Features or predicates? . . . 144
6.8.4 Inadequacy of vector dot-product models of similarity . . . 144
6.8.5 Scaling to have more episodes in memory . . . 145
6.8.6 Chunking, scaling, and networks of HRRs . . . 145
6.8.7 Comparison to RAAMs and tensor-product models . . . 146
6.9 Conclusion . . . 147

7 Discussion **149**
7.1 Transformations without decomposition . . . 149
7.2 Chunks and the organization of long-term memory . . . 151
7.2.1 The organization of long-term memory . . . 151
7.2.2 The opacity of chunks . . . 152
7.3 Convolution, tensor products, associative operators, conjunctive coding, and structure . . . 152
7.4 Implementation in neural tissue . . . 154
7.5 Weaknesses of HRRs . . . 155
7.6 Conclusion . . . 156

A Means and variances of similarities between bindings **158**

B The capacity of a superposition memory **160**
B.1 Scaling properties of a simple superposition memory . . . 160
B.1.1 Computation involved . . . 161
B.2 A superposition memory with similarity among the vectors . . . 163
B.2.1 The effect of the decision process on scaling . . . 164
B.3 Limitations of superposition memories . . . 167

C A lower bound for the capacity of superposition memories **168**

D The capacity of convolution-based associative memories **171**
D.1 A memory for paired-associates with no similarity among vectors . . . 171
D.2 A memory for paired-associates with some similarity among vectors . . . 174
D.3 Comments on the variable-binding memory with similarity . . . 177

E A lower bound for the capacity of convolution memories **179**

F Means and variances of a signal **181**

G The effect of normalization on dot-products **183**
G.1 Means and variances of dot-products of vectors with varying similarity . . . 183
G.1.1 Experimentally observed means and variances of dot-products . . . 184
G.2 Means and variances of dot-products of convolution expressions . . . 184

H HRRs with circular vectors **186**
H.1 Means and variances of dot-products of vectors with varying similarity . . . 186
H.2 Means and variances of dot-products for similarity and decoding . . . 188
H.3 Results on analogical similarity estimation . . . 189

I Arithmetic tables: an example of HRRs with many items in memory **191**
I.1 The objects and relations . . . 191
I.2 Queries to the memory . . . 192
I.3 Using fast estimates of the dot-product . . . 192

Bibliography **194**

List of Figures

1.1 Two ways of mapping a conceptual structure to units 14
1.2 A network expanding a reduced description to a full representation 15

2.1 Waltz and Pollack's network for parsing and interpretation 20
2.2 Components of Selman and Hirst's network for parsing sentences 21
2.3 Selman and Hirst's parsing network 22
2.4 A network constructed by ACME 25
2.5 Two schematic representations of the connections in Hinton's triple memory 27
2.6 Implementation of inheritance and exceptions in Hinton's triple memory . . 28
2.7 Converting from a local to a distributed representation 30
2.8 Family trees 31
2.9 Hinton's "Family trees" network 31
2.10 A network which implements a conjunctive code for colour and shape . . . 33
2.11 Conjunctive codes for 'red-square' and 'blue-circle' bindings 34
2.12 Decoding the superimposed bindings of 'red-square' and 'blue-circle' 35
2.13 DUCS architecture 37
2.14 Architecture of Sejnowski and Rosenberg's NETtalk system 38
2.15 A recurrent network, as described by Rumelhart, Hinton and Williams . . . 39
2.16 The type of recurrent net used by Elman 41
2.17 Elman's grammar which allows embedded clauses 42
2.18 Elman's recurrent net for predicting words 43
2.19 Trajectories of hidden units during processing 44
2.20 St. John and McClelland's sentence processing network. 45
2.21 Servan-Schreiber *et al*'s grammar and recurrent network 46
2.22 A binary tree and the triples which represent it 49
2.23 A Recursive Auto-Associative Memory (RAAM) for binary trees 50
2.24 A simple tree and the auto-associations that encode it in a RAAM 50
2.25 The outer product of two vectors 56
2.26 Convolution represented as a compressed outer product for $n = 3$ 56

3.1 Circular convolution represented as a compressed outer product for $n = 3$. 64
3.2 Circular correlation represented as a compressed outer product for $n = 3$ . . 65
3.3 Probability density functions for signals in a linear superposition memory . 69
3.4 A hetero-associator machine 70
3.5 The convolution of 8-dimensional vectors x and y in the frequency domain . 81
3.6 The exact and approximate inverses of a vector in the frequency domain . . 82
3.7 A convolution operation in a backpropagation network. 85
3.8 Bitwise dot-products versus floating-point dot-products 86

3.9 Construction and decoding of a HRR 94
3.10 Similarities of role-filler bindings from simulations 97

5.1 Example trajectory similarities 108
5.2 Subsystems to encode and decode a sequence 109
5.3 A Holographic recurrent network 110
5.4 A simple recurrent network 113
5.5 Performance of HRNs with varying number of hidden units 115
5.6 Performance of HRNs with varying numbers of symbols 115
5.7 Performance of various types of networks 116
5.8 Outputs of a HRN trained to produce trajectories through continuous space 118
5.9 A two-level system for decoding nested sequences 120

6.1 Two-stage models of analogy retrieval 124
6.2 The HRR for the probe is the weighted sum of binding chains 134
6.3 The binding chains in the contextualized HRR for the probe 139

7.1 A disordered compression of the outer product of x and y 155

B.1 Scaling of n with k in a simple superposition memory 161
B.2 Scaling of n with m in a simple superposition memory 162
B.3 Scaling of n with Pr(Error) in a simple superposition memory 162
B.4 Pdf's for signals in an superposition memory with similarity among vectors 165
B.5 Scaling of n with α in a superposition memory with similarity 165
B.6 Scaling of n with k in a superposition memory with similarity 166

D.1 Scaling of n with k in a paired-associates convolution memory 173
D.2 The different signals in a variable-binding memory with similarity 175
D.3 Scaling of n with k in a variable-binding memory with similarity 176
D.4 Scaling of n with α in a variable-binding memory with similarity 177
D.5 Scaling of n with m_p in a variable-binding memory with similarity 178
D.6 Scaling of n with $m - m_p$ in a variable-binding memory with similarity . . . 178

G.1 Variances of the dot-product of x with vectors similar to x 184

H.1 Means of the dot-product of a vector with vectors similar to it 187
H.2 Variances of the dot-product of a vector with vectors similar to it 187

List of Tables

2.1 Numbers of different sentences and patterns generated by Elman's grammar 43
2.2 A example receptive field table for a unit in the memory of DCPS 48

3.1 Means and variances of dot-products of common convolution expressions . 79
3.2 Base vectors 90
3.3 Token and role vectors 90
3.4 Similarities among some of the tokens 91
3.5 Sentences. 91
3.6 HRR frames 92
3.7 Similarities among the frames 92
3.8 Results of extracting fillers from the frames 93
3.9 Expected similarity of role-filler bindings 96

4.1 Correspondences between the circular and standard systems 103
4.2 Means and variances of dot-products for decoding and comparing bindings 104
4.3 Means and variances for comparing vectors 104

5.1 Direct and recursive methods of decoding a trajectory-associated sequence . 108

6.1 Shape-configurations 126
6.2 HRRs for some of the shape configurations 127
6.3 The probe episode and the memorized episodes 128
6.4 Similarity scores from fast and slow similarity estimators 131
6.5 Base vectors and token vectors 132
6.6 Results from Experiment 1 132
6.7 Results from Experiments 2 and 3 139
6.8 Interpretation of an analogy: extraction of corresponding entities 141
6.9 Criteria for good analogies in SME and ARCS 143
6.10 Approximate scores from ordinary and contextualized HRR comparisons . 147

7.1 Vector-space multiplication operations 153

A.1 Means and variances of the dot-products of the bindings 159

B.1 Means and variances of signals for a superposition memory with similarity 164

D.1 Means and variances of signals in a paired-associates convolution memory 172
D.2 Classes of values and variables in a variable-binding memory with similarity 175
D.3 Means and variances of signals in a variable-binding memory with similarity 176

G.1 Experimentally observed statistics of dot-products of common convolution expressions 185

H.1 Means and variances of dot-products for decoding and comparing bindings 189
H.2 Results from Analogy Experiment 3 for standard and circular vectors 190

I.1 CPU times from runs using fast bitwise comparisons 193

Chapter 1

Introduction

Since the early 1980's there has been considerable interest in connectionist models for higher-level cognitive tasks such as natural language processing and reasoning. Many models have been proposed and built. The success of these models has been mixed – most appear to have severe limitations.

In connectionist models, even more than in other AI models, the representations chosen for the data determine what the system can do, since the processing is generally simple. Consequently, if we want to understand the difficulties and possibilities of applying connectionist models to higher-level tasks, it is appropriate to focus on the types of representations that are and can be used in models of such tasks.

Much of the interest in connectionist models for higher-level processing stemmed from dissatisfaction with the limitations of symbolic rule-based systems. These include brittleness, inflexibility, difficulty of learning from experience, poor generalization, domain specificity, and sloth of serial search in large systems. Initial results with connectionist models seemed to suggest potential for overcoming these limitations. Various connectionist models have tantalized researchers with many attractive properties: pattern completion, approximate matching, good generalization, graceful degradation, robustness, fast processing, avoidance of sequential search, parallel satisfaction of soft constraints, context sensitivity, learning from experience, and excellent scaling to larger systems.

However, no single connectionist model has had all these properties and researchers have found it very difficult to perform higher-level reasoning tasks in connectionist models. The difficulties can in part be traced to two characteristics of higher-level reasoning tasks. The first is that the temporary data structures required for higher-level reasoning are often complex, and cannot be represented in common connectionist representation schemes. The second is that many higher-level reasoning problems appear to at least sometimes require sequential processing, and connectionist models lack the procedural controls necessary to control sequential processing.

The focus of this thesis is on how some of the structures that are required for higher-level reasoning can be represented in a distributed fashion. In this introduction, I first discuss issues of representation. Next I summarize the advantages and disadvantages of local and distributed representations. Finally, I review Hinton's [1987; 1990] influential notion of a "reduced description", which provides ideas and desiderata about the representation of hierarchical structure in connectionist models.

I assume that the reader is familiar with the basic principles of connectionist models. For a tutorial introduction, I refer the reader to McClelland and Rumelhart [1986].

1.1 Representations, descriptions, and implementations

In this thesis I am more concerned with how complex data structures can be implemented elegantly in distributed representations than with the choices involved in devising a good knowledge representation for a particular task, though I do approach the latter in Chapter 6.

In contrast to Fodor and Pylyshyn [1988], but in common with Hinton, McClelland and Rumelhart [1986], I do see implementation issues as interesting and important because implementation choices often determine which operations are fast and easy to compute and which are slow and difficult. Chalmers [1990] and Niklasson and van Gelder [1994] note that connectionist representations for compositional data structures are especially interesting because they raise the possibility of content-sensitive manipulation without decomposition, which appears to have no analogue in conventional symbol-based representations.

Some authors use the word "representation" to refer to a scheme for representing a class of entities, and "description" to refer to the way a particular entity is described. I use "representation" to refer to both representation schemes and descriptions of particular entities – the context should make the intended meaning clear.

1.2 Connectionist models

A connectionist model is typically a fixed network of simple processing units. The units are connected by weighted links and communicate only via these links. The messages sent along the links are typically scalar values, which are modulated by the weight on the link. Units compute some simple function of the input they receive, such as the sigmoid of a weighted sum. The value of the function is the state or activation of the unit, and this is the message passed on to other units. In nearly all connectionist models, units can (in theory) compute their values in parallel, and the overall computation time is short – on the order of 100 steps or less.

There are two distinct locations for representing knowledge (or more neutrally, data) in connectionist models. The first is in the activation values of the units. These are typically used for representing short-term data; the particular instance of the task, intermediate states in the computation, and the result. The second location for representing knowledge is the weights on the connections among units. These comprise the long-term memory of the model and are typically used for representing knowledge about the domain. They can also be used to store particular episodes – the weights can be set so that they can later reinstantiate a particular activity pattern over the units.

There are really only two ways in which links can be used to encode domain knowledge. One is as constraints on solutions to a task, the other is as transformations between input and output patterns. Constraints are often used with those localist models that process by settling to a stable or low-energy state. These states are (or are hoped to be) those in which most constraints are satisfied. In the network the constraints represented by a link are very simple: either the connected units should be active together (an excitatory link), or if one is active, the other should inactive (an inhibitory link). Constraints are generally soft, in that they can be overridden – the importance of a constraint is indicated by the weight on the link. Pattern transformations are generally used in feedforward and recurrent networks, and are usually learned from examples. Some networks, such the Hopfield [1982] network (an error-correcting associative memory) can be interpreted in both ways.

There is far more diversity of representation schemes for the temporary data structures necessary to solve a given instance of the task. The components of the task can be sentences, words, letters, phonemes, features, concepts, visual or auditory signals, measurements, etc. These components must be represented as activations over the units of the network. To a large extent, the representation chosen for the task constrains how the knowledge is encoded in the links and determines what the network can compute.

1.3 Representational issues

Representations are of critical importance in connectionist models. To a large extent, the representation scheme used determines what the system can compute. Representations are the source of both strengths and weaknesses of connectionist models. Carefully constructed connectionist representations can endow systems with useful properties such as robustness, ability to learn, automatic generalization, graceful degradation, etc. These properties tend to be more associated with distributed representations than with local representations (though local representations are not without uses). However, there are serious problems with the adequacy of distributed representations for storing the type of information necessary to perform complex reasoning.

In this section I give an overview of issues that arise with connectionist representations. The issues concerning representational adequacy, computational properties, scaling, and suitability for learning are central to this thesis. There are other important but less pressing issues which I mention here but do not return to.

1.3.1 Representational adequacy

A system will not be able to perform complex reasoning tasks if it cannot even represent the objects involved in the tasks. For many complex tasks it seems necessary to be able to represent complex structures, such as hierarchical predicates. This is an issue especially for models using distributed representations. When distributed representations first began to catch people's interest there was no satisfactory way of representing complex structure with them, and although several methods have been proposed since, all have had serious drawbacks. The difficulty with representing complex structure was highlighted by Fodor and Pylyshyn [1988] in their strong and influential criticisms of connectionist models.

The main categories of structures that complex reasoning seems to require are: variable bindings, sequences of arbitrary length, predicate structures, and recursive predicate structures. Some sort of variable binding is required anywhere that general rules are used. Systems which deal with sequential input, such as written or spoken language, must have some way of storing relevant aspects of the input, since the appropriate output may depend upon input encountered some time previously. For example, in speech understanding, the appropriate interpretation of vowel sound can depend on the preceding sounds. Often, the amount of input that must be stored is unknown. The ability to represent predicate (relational) structures is necessary in systems which deal with input in which relationships among the entities are important. In tasks like language understanding, the relationships can be hierarchical, leading to the need to represent hierarchical predicates. For example, the object of a predicate such as 'think', 'see', or 'want' can be another predicate.

Connectionist representations are often very different from conventional data structures, such as variables, records, lists and trees. Sometimes the representations are nearly

unintelligible and appear to have little or no relationship to conventional data structures. However, if the model can be proved to work correctly, then it must be representing the information required for the task in some fashion. Often, this can be analyzed in terms of conventional data structures. Furthermore, when designing a connectionist model to perform some task, one must consider what sort of structures the model is going to represent and how it will manipulate them. When designing or analyzing connectionist representations for complex structures, a number of questions concerning several aspects of representation and processing arise:

- Composition, decomposition and manipulation: How are components composed to form a structure, and how are components extracted from a structure? Can the structures be manipulated using connectionist techniques?
- Productivity: A few simple rules for composing components can give rise to a huge variety of possible structures. Should a system be able to represent structures unlike any it has previously encountered, if they are composed of the same components and relations?
- Immediate accessibility: What aspects of components and structures does the representation make explicit and accessible without pointer-following?
- Systematicity: Does the representation allow processes to be sensitive to the structure of the objects? To what degree are the processes independent of the identity of components of structured objects? (The term 'systematicity' comes from Fodor and Pylyshyn [1988].)

There have been three broad classes of reactions to the problem of representing complex structure. The first is exemplified by the argument of Rumelhart and McClelland's [1986b] that the importance of the ability to compose and decompose complex structure is overrated. The second is exemplified by the work of Smolensky [1990] on designing connectionist representations for complex structure. The third is exemplified by the work of Pollack [1990] and Elman [1990] on connectionist networks which learn representations for complex structure. In this thesis, I take the same approach as Smolensky, because I believe compositional structure is important, and because the properties of designed representations are easier to analyze and understand than those of learned representations.

1.3.2 Computational properties

The computational properties of a representation relate to what can be computed using the representation, the simplicity and speed of the computation, and the resources required. Connectionist networks usually have very simple processing, so if a representation does not make something simple to compute, it most likely cannot be computed by the network at all.

Computational properties commonly possessed by connectionist representations, and often cited as reasons for interest in connectionist models, include the following:

- Immediate accessibility (explicitness): information can be extracted with simple processing and without further access to memory.

- Analogical representation: similar objects have similar representations.[1]
- Representational efficiency: representational resources are used in an information-efficient manner.
- Affordance of generalization: the representation supports generalization of knowledge.
- Robustness and graceful degradation: the information in the representation is still usable when parts are corrupted or missing.
- Affordance of error correction and pattern completion: the representation contains redundant information which makes this possible.
- Affordance of easy associative access: the representation make associative access simple.

This list is not, and cannot be, complete, because it is always possible to invent new representations with novel computational properties. The properties in this list are interrelated, and some can be considered as superclasses or prerequisites of others.

The most important of these properties is immediate accessibility. Information in a representation immediately accessible if it can be extracted with little computation and without further access to memory (as would be entailed by pointers). Accessibility, or explicitness, is really a matter of degree: information that requires only a little computation to extract is somewhat explicit, but information which requires extensive computation to extract is non-explicit. Almost all connectionist representations are explicit – by necessity, since the processing they use is so simple.

One type of explicitness that is ubiquitous (and very useful) in distributed representations is explicit representation of similarity. Similarity is explicit when the information of which similarity is judged is immediately accessible. This type of explicitness is one of Marr's [1982] desiderata for representations of 3-D objects, which specifies that representations should be analogical. By this he means that representations should in some sense be analogues of the objects they represent, and should reflect their similarity. Another of Marr's desiderata, which complements that of having analogical representations, is that representations should make differences between similar objects explicit. This property has been discussed little in connectionist work to date, but will probably become important as connectionist models grow more sophisticated.

I will have more to say on the other computational properties in the next section, where I discuss local and distributed representations.

1.3.3 Scaling

Connectionist models are often demonstrated on toy problems which involve only a small fraction of the entities and relationships found in real-world problems. Hence, the way in which the resource requirements of a representation scale with increasing numbers of entities and relationships is important. The way in which increasing size of representation affects the rest of the system must also be considered. For example, if the number of components in a representation scales linearly with the number of different objects, and the system involves interactions among all pairs of components of the representation, then the scaling of the entire system is quadratic.

[1] In connectionist systems, the vector dot-product (inner-product) is often used as a measure of the similarity of two representations.

1.3.4 Learning representations for new concepts

One of the main attractions of connectionist models is their ability to learn. This is related to scaling: while it is possible to fashion by hand representations for toy problems, it would be very time consuming and difficult to fashion representations for real-world problems. Consequently, the suitability of a representation for use with learning algorithms is important.

1.3.5 Other representational issues

There are a number of philosophical issues which arise in symbolic artificial intelligence research, and which also are relevant to connectionist research. However, connectionist models have not, for the most part, reached the level of sophistication where these issues become crucial. Hence, I mention these issues here, but do not present any detailed discussion of them. There are also other technical issues concerning meta-level reasoning and procedural control, which I mention here but not elsewhere for similar reasons.

Sophistication of knowledge representations

Most connectionist models have extremely simple knowledge representation schemes that do not make strong distinctions between ontological categories such as concepts, roles, classes, and individuals. Whether or not the lack of these distinctions is a problem remains to be seen. Some connectionist representations do support abilities such as property inheritance, without marking entities as concepts, subconcepts, or individuals. Generally, these abilities arise from analogical properties of the representations.

The Knowledge Representation Hypothesis

The Knowledge Representation Hypothesis [Brachman 1985b] is an explicit statement of an idea that has guided much work in symbolic artificial intelligence. It states that intelligence is best served by representing knowledge explicitly and propositionally. Connectionist researchers would, for the most part, appear not to believe in this hypothesis. In some connectionist models knowledge about the domain is represented propositionally, e.g., in Shastri's [1988] and Derthick's [1990] models, but this knowledge is compiled into the links of the network and the processing does not access the propositional representations. Most connectionist researchers appear to take the view that if the relevant aspects of the particular task are represented suitably, then fast processing can be accomplished with knowledge represented implicitly in the links of the network.

Logical soundness and completeness vs. computational tractability

Connectionist models do not escape the tradeoff between computational tractability and logical soundness and completeness [Levesque and Brachman 1985]. However, computational speed has always been important to connectionist researchers, and most models are built to provide fast answers, but with no guarantee of correctness. Shastri [1988] is

one exception; he has investigated what types of limited reasoning can be performed both soundly and correctly.

Meta-level reasoning

Almost all connectionist models make a strong distinction between long-term knowledge about how to perform the task, which is stored in the weights on the connections, and the short-term knowledge about a particular instance of the task, which is stored in the unit activations. It would seem impossible to reason about the rules, because they are represented in the weights and cannot be (at least directly) transferred into the activations. However, some other types of meta-level reasoning, e.g., monitoring the progress of a long chain of processing, are possibly very useful. A way to do this might be to have one network monitor the activity of another.

Procedural control

Connectionist models which perform higher-level reasoning such as following chains of inferences will undoubtably require some form of procedural control, possibly involving subroutines. Ideally, the behaviour of the control mechanism should be error-tolerant and generalize to novel situations. Very little work has been done on this problem.

1.4 Connectionist representations

There are two distinct styles of representation used for short-term data in connectionist models: local and distributed representations. They appear to have quite different properties but are best regarded as end-points of a continuum. I am most interested in distributed representations, but, as there is some insight to be gained from considering local representations, I will discuss both in this introduction. The main topic of the remainder of this thesis is how complex structure can be embodied in distributed representations in a way which preserves the attractive properties of distributed representations.

1.4.1 Local representations

In a local representation concepts (objects, features, relationships, etc) are represented by particular units. For example, the "celestial body" sense of "star" is represented by a single unit in Pollack and Waltz's model of word sense disambiguation. Activity on this unit represents the "celestial body" meaning of the word "star", as opposed to the "movie star" sense. Local representations are commonly used in constraint satisfaction and in spreading activation networks.

Advantages of local representations

The advantages of local representations mostly relate to explicitness and ease of use. Some prominent advantages are the following:

- Explicit representation of the components of a task is simple.
- The interface for the experimenter is convenient.

- It is easy to represent a probability distribution over different possibilities.
- It is relatively easy to design representational schemes for structured objects.

The principal virtue of localist representations is that they make all the possible components of a solution explicit. A unit represents a component, and activity on the unit represents the presence of that component in the solution. Global constraints on solutions are transformed into a set of simple local constraints, each involving two of the possible components of the solution. These simple constraints can be reinforcing or inhibiting; the presence of one component encourages or discourages the presence of another component in the solution. Each simple constraint on how components interact is implemented by an excitatory or inhibitory link. The constraints are usually symmetric: if A and B are linked, then A excites or inhibits B to the same degree that B excites or inhibits A. Converting constraints to simple local constraints can require the creation of more units. For example, if the existence of a relationship between two entities depends on the existence of another relationship between two other entities, then units must be assigned to these relationships. This can lead to large numbers of units for even simple problems. Once all the interacting components are explicitly denoted by units, and constraints embodied in links, reasonably good solutions can be found using simple computation. If links are symmetric, the computation can be viewed in terms of energy minimization (as in Boltzmann machines [Ackley, Hinton and Sejnowski 1985] and Hopfield networks [1982]).

Local representations are convenient to use for input and output to a network because they are simple and unambiguous and can be easily compared against desired results.

Presenting output as a probability distribution over possibilities is very useful for tasks which involve prediction or classification. This requires using activation values between zero and one and constraining the activations of a set of units to sum to one. The units must represent a complete set of mutually exclusive possibilities.

Problems with local representations

The main problems with local representations concern inefficiency and learnability:

- Inefficiency of local representations for large sets of objects.
- Proliferation of units in networks which represent complex structure.
- Inefficient and highly redundant use of connections.
- Questions about the learnability of elaborate representations for complex structure.

There are several ways in which local representations are inefficient. The most obvious way is that it is inefficient to use n units to represent n different objects. While it might be practical to use a hundred units to represent one hundred different objects, it is seldom practical to use a million units to represent one million different objects. Another type of inefficiency occurs with some localist representations for complex structure. When problems involve complex structure, the number of possible components of a solution tends to increase very quickly with the size of the problem. Representations which devote units to all possible components of a solution tend to require very large numbers of units for large problems. The third way in which local representations can be inefficient has to do with the use of connections. If A and B are units representing similar things, then A and

B will have similar interactions to other units. However, in a localist network, these similar interactions must be represented by independent duplicate sets of links. This can result in duplication and inefficient use of links. All of these inefficiencies lead to poor scaling properties.

The types of networks which researchers design to perform complex reasoning tasks are quite elaborate. Although various learning algorithms can be used with localist networks, none of these algorithms would be able to derive localist networks such as those I review in Chapter 2. Furthermore, the representations derived by connectionist learning algorithms which do build internal concepts, such as multilayer feedforward (backpropagation) networks, tend to be distributed rather than local.

1.4.2 Distributed representations

Hinton *et al* [1986] define a distributed representation as one in which each concept is represented over a number of units, and in which each unit participates in the representation of a number of concepts. The size of distributed representations is usually fixed, so that the representation can be instantiated over a fixed set of units. The units can have either binary or continuous-valued activations.

Hinton *et al* [1986] give a good general introduction to distributed representations. Anderson [1973] proposes and analyzes a psychological model for memorizing lists, which uses superimposed continuous distributed representations. Rosenfeld and Touretzky [1987] discuss binary distributed representations and analyze how many patterns can be superimposed before the individual patterns become unrecognizable. Van Gelder [1990] analyzes distributed representations from a more philosophical viewpoint and develops an alternative definition to that of Hinton *et al*.

In some distributed representations the individual units stand for particular features, like "is-red", and in others it is only the overall patterns of activity which have meaning. Hinton *et al* [1986] use the term "micro-features" to describe individual units which can be given an unambiguous meaning. To the extent that this can be done, these parts of the representations are like local representations.

Distributed representations are in many ways analogous to greyscale images or photographs. Distributed representations are vectors (one dimensional arrays) with many elements (units). Concepts are represented by patterns in this continuous, high-dimensional space. Greyscale images are two-dimensional arrays with many elements (pixels). (The two-dimensional nature of images is irrelevant to this analogy, what matters is that elements of a greyscale image take a scalar value, and have a fixed position.) Physical objects are represented by patterns in this fine grained, high-dimensional space. In both, the information content is distributed and redundant, which makes them robust and suitable for error correction. Many pixels in an image can be changed without making the image unrecognizable. Similarly, in a distributed representation some noise can be added to the activation levels of units without making the representation of an object unusable.

When viewing superimposed images (like multiple-exposure photographs) it is easy to tell apart the individual images if they are distinct and coherent. Similarly, distributed representations of several items can also be superimposed by adding the vectors together. It is easy to identify the various items in the resulting vector, provided that they are not too similar and the representation has sufficient capacity.

Advantages of distributed representations

The attractiveness of distributed representations is largely due to the ease of endowing them with the following computational properties:

- Explicit representation of relevant aspects of objects.
- Analogical representation, i.e., similar representations for similar objects (explicit similarity).
- Redundant storage of information.
- Efficient use of representational resources.
- Continuity, i.e., representation in a continuous vector space.

Explicitness is possible with distributed representations because they are wide and flat, which provides much opportunity for representing relevant aspects of objects. Explicit representation makes fast processing possible, because it is not necessary to chase pointers or perform deductions to get at the information in a representation.

A ubiquitous type of explicitness is that which makes similarity explicit – similar concepts have similar representations. The similarity of distributed representations can be defined as the Euclidean or Hamming distance between two representations, which can be computed quickly. Thus, the similarity of objects represented by distributed representations can be quickly computed.

Explicit similarity, the ability to represent similar objects by similar representations, is probably the most useful property of distributed representations. It can assist in the fast retrieval of similar instances from memory, and in matching against prototypes and rules. In Chapter 6, I investigate how far this idea can be taken – I look at whether distributed representations can support judgements of the structural similarity of structured objects.

Automatic generalization is another useful benefit of explicit similarity. The nature of processing in connectionist models, which involves units computing weighted sums of their inputs, results in systems that usually respond similarly to similar representations. A novel input that is similar to a familiar input will likely result in an output similar to that for the familiar input, which will often be an appropriate output for the novel input.

Information in a distributed representation is generally stored in a redundant fashion – we only need to see some part of the representation to know what the whole is. Because of this, distributed representations are robust in the presence of noise and degrade gracefully as noise is added. Redundancy makes pattern completion and error correction possible.

The ability to superimpose the representations of multiple items in the same set of units is another benefit of redundancy. Individual items can still be recognized, depending on the degree of redundancy and the number of representations superimposed. There is a soft limit to number of items that can be superimposed – representations degrade as items are added.

Distributed representations usually require far fewer units to represent a set of objects than do local representations. This is because the number of possible patterns of activation over a set of units is much greater than the number of units. At one extreme, a binary-number style representation is maximally efficient – there are 256 different patterns of ones and zeros over eight units. At the other extreme, a local representation is minimally efficient – eight units can represent only eight different objects.

The efficiency of a representation has consequences for the rest of the system. An inefficient representation can lead to duplication of computational machinery. For example, in a local representation, two units representing concepts which interact with other concepts in the same or a similar way will have many similar connections, which could likely be avoided through the use of a more efficient representation.

To some extent, efficiency is in opposition to redundancy (though inefficiency does not necessarily result in redundancy). However, in practice, a moderate number of units provides so many possible patterns that it is easy to make distributed representations sufficiently efficient and redundant.

If units have continuous activation values, then concepts are represented in a continuous vector space. One benefit of continuity is that it makes for natural representations of inherently continuous concepts, such as colour, the degree of danger presented by a wild animal, or the meaning of an utterance. This gives a system the potential for coping with such things as nuances of meaning and slight variations in context .

Another important benefit is that having a continuous representation space allows us to make small changes to representations. This means that continuous distributed representations can be used in neural networks that learn by gradient descent techniques. An appropriately constructed system can learn good distributed representations for the objects in its domain.

Problems with distributed representations

The main problems with distributed representations are the following:

- Difficulties with representing arbitrary associations and variable bindings.
- Difficulties with representing sequences of arbitrary length.
- Difficulties with representing predicates and hierarchical structure.
- Questions about the origin of patterns which represent particular objects.
- Difficulties with using distributed representations for input and output to the network.

The main problems with distributed representations concern representational adequacy. Work on models for complex tasks has been held back by difficulties with representing complex structure.

In many types of distributed representations, patterns can be superimposed to indicate the presence of several different objects. However, this does not represent associations between the objects. Furthermore, crosstalk and illusory conjunctions can occur. For example, if we store 'red circle' and 'blue square', by superimposing the representations of 'red', 'circle', 'blue', and 'square', it will appear as though 'red square' was also stored. The same problems arise with variable binding.

Storing variable-length sequences in distributed representations is difficult, because both the identity and the order of inputs can be important and must be remembered. One of the most common techniques for dealing with time varying input is to turn time into space, which is done by having a set of buffer units for storing the input for some fixed number of timesteps in the past. As each new input is received, previous inputs are shifted back in the buffer. This is somewhat inelegant and is limited in that the network only remembers inputs for however many time-slices there are in the buffer – input which is pushed out of the end of the of the buffer is irretrievably lost. A different approach is to use recurrent networks (Section 2.3.2) to try to learn what and how much input must be remembered.

A distributed representation of a predicate such `bite(spot,jane)` (for "Spot bit Jane") must be carefully designed to preserve the information about which entity is associated with which role. An obvious way to avoid ambiguity about who is the agent ('Spot') and who is the object (or patient, 'Jane') is divide the representation into several blocks, and devote a block to each role. However, this method is unsuitable for representing recursively nested predicates such as "Spot bit Jane, causing Jane to flee from Spot": `cause(bite(spot,jane),flee(jane,spot))`, because the representation of a predicate is larger than the representation of an object, and cannot be squeezed into the same set of units. The difficulty is not so much with representing the objects involved (we could simply superimpose them) as it is with representing the relationships among the objects in an unambiguous and explicit fashion.

Another concern with distributed representations is how the patterns are obtained. Researchers have generally used one or a combination of the following three methods: hand-design, random generation, or learning. In many of the early connectionist systems, the representations were designed by hand, using units to code for features of objects. This made it easy to introduce and control explicit similarity among representations, which often underlie the interesting properties of these systems. However, hand-design tends to be unsatisfactory for several reasons: it is very laborious to construct representations for each entity a system deals with, sometimes the relevant features of the objects are not known, distributed representations are often so wide that there are many more units than features, and finally, an intelligent system should be able to develop its own representations.

Another method for obtaining patterns is to use random patterns. This is done in many psychological models, e.g., those of Anderson [1973], and Murdock [1982]. In Chapter 3, I describe how this method can be combined with feature-based methods in order to introduce explicit similarity among representations.

The most promising and interesting method for obtaining patterns is to have the system learn appropriate representations for the given task. Hinton showed that this could be done in a feedforward network trained using backpropagation, I describe this and related methods in Section 2.2.2.

Distributed representations are usually not suitable for input and output to a network, because they do not provide a single value which indicates to what degree some object is present in the representation. This is not a big problem, but it is usually necessary to translate between distributed and local representations for input and output. This can require significant amounts of computation, especially for translating from distributed to local representations.

1.4.3 Relationships between local and distributed representations

The definitions of local and distributed representations are matters of degree. Representations in which many units are active are fully distributed, those with just a few units active are somewhat local and are sometimes called sparse distributed representations, and those with just one unit active are fully local. Furthermore, whether or not we consider a representation to be local or distributed depends on which level at which we examine it. The representation of the lowest-level concepts or features in a model may be clearly "local"

while the representation of concepts composed of those features may have the characteristics of a distributed representation. One such system is Derthick's [1990] μKLONE (Section 2.1.2).

Smolensky [1986] discusses transformations between local and distributed representations, and the conditions under which operations on both are equivalent. Transforming representations of a set of objects from a local representation to a distributed one is simple. It usually just involves superimposing the distributed representations of the objects present in the local representation. If the superposition operation is linear (as opposed to a non-linear threshold operation), then this transformation from local to distributed representations is linear. A system which performs linear operations on the distributed representation is equivalent to one which performs linear operations on the local representations. If the distributed representations for all objects are linearly independent, then there is a linear transformation from the distributed to the local representation. However, requiring linear independence of patterns is inefficient, as there are only n linearly independent patterns over n units.

Some representations like the binary system for numbers satisfy the definition of a distributed representation, but there is a common sentiment that they are not distributed representations (e.g., van Gelder [1990]). I would claim they are distributed representations, but without most of the useful properties that distributed representations make possible. The aspect of numbers which binary representations make explicit is the powers of two whose total equals a number. This is useful for logic circuits that add and multiply numbers, but it is not very useful for anything else. In particular, the similarity structure this induces on numbers has little relationship to any human judgements of similarities of numbers.

1.5 Reduced Descriptions

Hinton [1987; 1990] originated the idea of a "reduced description" as a method for encoding complex conceptual structure in distributed representations. Reduced descriptions make it possible to use connectionist hardware in a versatile and efficient manner. According to Hinton, there should be two ways of representing a compositional concept (i.e., a structured object whose parts are other, possibly compositional, objects). One way is used when the concept is the focus of attention. In this case all the parts are represented in full, which means the representation of the concept could be several times as large as that of one of its parts. The other way of representing the concept is a reduced description, which is used when the concept is not the focus of attention, but is a part in another concept. In this case the representation of the concept must be the same size as that of other parts. Concepts are taken to be simple frame-like structures, with a number of roles or slots. Relations (predicates) can be represented by allowing one role to be the relation name and the other roles to be the arguments of the relation. The two different ways a concept can be represented in the system are shown in Figure 1.1. The concept D, which has E, F, and G filling its roles, can either be the focus of the system, or can be a filler in the representation of the concept A. A reduced description is like a label or pointer for a concept, to be used when we want to refer to the concept, but not focus on it. Since reduced descriptions provide a pointer-like facility, they allow us to represent concepts linked in an arbitrary tree or graph structure.

Figure 1.2 shows the steps involved in accessing a full representation from a reduced description. To begin with, the full concept A is instantiated in the network. The reduced

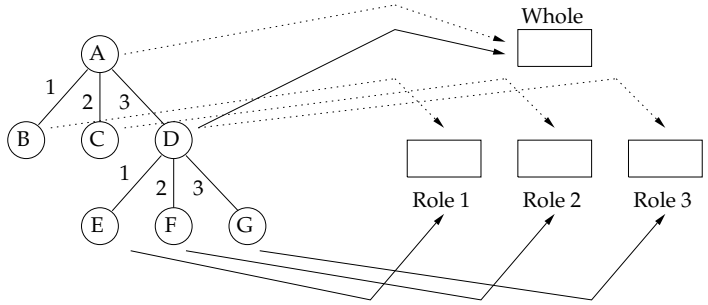


Figure 1.1: Two ways of mapping a conceptual structure (the tree on the left) to the units in a connectionist system. In the first way (the solid arrows) the concept D is the focus of attention, and the fillers of its roles are mapped to the appropriate sets of units. In the second way (the dashed arrows) A is the focus of attention and just the reduced description for D is mapped to a set of units. The rectangles labelled 'Whole', 'Role 1', etc, are sets of units over which a distributed representation can be instantiated. (Adapted from Hinton [1990].)

description of concept D is the filler of its third role. To access the full representation of D we first transfer D into the focus position, and then expand the reduced description of D to get its full representation.

This scheme for mapping structures to hardware is versatile and efficient because it allows many different concepts to be represented on the same hardware. An alternative is to have hardware dedicated to each different type of concept, but this can require excessive amounts of hardware and cannot easily represent nested structures.

Hinton's description of reduced descriptions involve four desiderata, all of which seem to be essential for any useful distributed representation of structure:

- Representational adequacy: it should be possible to reconstruct the full representation from the reduced description.
- Reduction: the reduced description should be represented over fewer units than the full representation.
- Systematicity: the reduced description should be related in a systematic way to the full representations.
- Informativeness: the reduced description should tell us something about the concept that it refers to, without it being necessary to reconstruct the full representation.

The necessity of the first desideratum is self-evident – if the reduced description does not provide enough information to reconstruct the full representation, then it is not an adequate representation. The second desideratum comes from practical considerations of working with a fixed-size set of computational units. Furthermore, the precision with which activation values must be represented should not be greater for the reduced description than for the full description – it is possible, but not realistic, to store an infinite amount of information in an infinite precision number. This means that the reduced description must have less information about components than the full description – some information must be thrown away in constructing the reduced description. This ties in

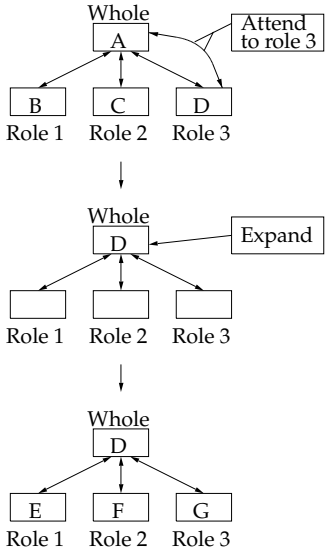


Figure 1.2: A network focusing on a reduced description (D) in another concept (A) and expanding it to a full representation (E F G). (Adapted from Hinton [1990].)

with Miller's [1956] notion of a chunk and his claim that people can store "seven plus or minus two" chunks in short-term memory. The idea of "recoding" a set of items (which are themselves chunks) into one chunk is central to Miller's explanations of how human short-term memory works. A reduced description is equivalent to a chunk, and a full representation is equivalent to the contents of the chunk.

The third and fourth desiderata are for something more than that supplied by conventional pointer-based ways of representing hierarchical structure. Pointers are generally arbitrarily related to the contents of the structure they point to. Having a systematic relationship between label and structure makes it easier to find one given the other. The desire for informativeness is for a form of representational explicitness – the reduced representation should make explicit some information about the concept it refers to.

Pointers in conventional random access memory serve the same structural function as reduced descriptions. A record corresponds to a concept, and fields in a record are the roles. The contents of the fields are the fillers. The entire record is the full representation of a concept, and the pointer to the record is the reduced description. Complex structure is represented by allowing fields to be pointers to other records. If we have a pointer, we can reconstruct the full concept by following the pointer (i.e., by accessing the memory). However, without the use of some sort of indexing, it is difficult to find the pointer to a structure given its contents.

The difference between pointers and reduced descriptions is that the relationship between a pointer and what it points to is arbitrary. Pointers tell us nothing about what they point to – the only way we can find anything out about the record pointed to, or its fillers,

is to follow the pointer.[2] Hash indexing is a technique from conventional data-structure theory which is close in spirit to reduced representations. It provides a way of deriving the "label" of a full representation given its contents. However, memory lookup is usually necessary to find the contents given the label.

The chief potential advantage of reduced descriptions over pointers is that they could provide more explicit labels for concepts. This could make it possible to process concepts more efficiently, since it would not always be necessary to follow pointers to discover information about subconcepts.

A reduced representation can be seen as a compressed version of a full concept. The compression must be invertible, since we need to be able to reconstruct the full concept from the compressed version.[3] Unfortunately, Hinton [1990] did not have any way of implementing reduced descriptions, he only described the properties reduced representations should have. Pollack [1990] used backpropagation to learn reduced descriptions for tree structures (Section 2.4.2). In Chapter 3, I describe a way of constructing reduced representations from first principles.

1.6 Discussion

The computational properties which can be built into distributed representations make them worth considering as a representation for higher-level cognitive tasks. In Chapter 2, I review various connectionist models that are relevant to the representation and processing of complex structure, paying particular attention to representation schemes.

In Chapter 3, I present and discuss a representational scheme for hierarchical structure, called "Holographic Reduced Representations" (HRRs), which I believe overcomes some of the flaws of other representations. HRRs use a simple scheme for building up representations of structures from representations of components, and are amenable to analysis. HRRs support the useful properties of distributed representations that were identified in Section 1.4.2: explicitness, redundancy, continuity, and efficiency. In Chapter 4, I describe how HRRs can be implemented with distributed representations based on complex-valued numbers. This makes some of the operations faster to compute, and makes it possible to perform an advantageous type of normalization.

In Chapter 5, I describe how the convolution operation, the basic association mechanism for HRRs, can be incorporated into recurrent networks. This allows representations for items and arbitrary length sequences to be learned using gradient descent procedures. I claim that for the purposes of storing sequences in a 'context' layer (à la Elman [1990], described in Section 2.3.2) this type of network is superior to a standard recurrent network.

In Chapter 6, I investigate the extent to which the explicit similarity of HRRs is related to useful forms of similarity. It is obvious from first principles that HRRs will induce some sort of similarity structure on structured objects, since convolution is a similarity-preserving operation. The question is whether the induced similarity structure is useful or related to any other commonly used measures of similarity. I discuss Gentner and

[2] Except in "tagged" memory hardware, such as that used in some LISP machines. In these systems, several address bits are used to indicate the type of the object at that address, e.g., atom or cons cell. This helps to increase the speed of LISP programs.

[3] MacLennan [1991] claims this is impossible to do with continuous representations, due to a theorem of Brouwer's which states that Euclidean spaces of different (but finite) dimensions are not homeomorphic. This means that it is not possible to have a continuous function $f : E^n \times E^n \rightarrow E^n$ with a continuous inverse $f^{-1} : E^n \rightarrow E^n \times E^n$.

Forbus's [1989] notions of the analogical similarity of structured objects. Using a set of simple examples, I show how the similarity structure induced by HRRs captures some aspects of analogical similarity, and how including other types of associations in addition to role-filler associations can help to capture more aspects of analogical similarity.

In Chapter 7, I discuss various issues: how HRRs can be transformed without decomposition; differences between HRRs and some psychological notions of chunks; how other vector-space multiplication operations could be used instead of convolution; how a disordered variant of convolution might be implemented in neural tissue; and weaknesses of HRRs.

Chapter 2

Review of connectionist and distributed memory models

In this chapter, I review various connectionist models and distributed memory models that are relevant to performing higher-level tasks. I begin with some localist models for higher-level reasoning, and then describe some distributed models which deal with simple structures. Following that I discuss some models which have dealt with problems of representing hierarchical structure in implicit ways, by trying to get relatively simple recurrent networks to learn to process language. Finally, I discuss some models and representation schemes which are explicitly designed to represent hierarchical structure. The selection of models for this review was made on the basis of how interesting, novel, or influential the representation scheme is. The primary goal of this review is to discuss the advantages and disadvantages of various connectionist representation schemes, so I tend to ignore aspects of these models that do not have much to do with representational issues. The secondary goal is to familiarize the reader with the range of connectionist approaches to higher-level cognitive tasks, and the limitations that many of these approaches share, the most prominent of which are difficulties with scaling up to larger than toy problems. The reader who is only interested in models closely related to HRRs should just read Section 2.2.3 and from Section 2.4.2 onwards.

2.1 Localist connectionist models

Localist networks that compute by spreading activation and local inhibition were among the first connectionist models of higher-level tasks. A unit in one of these networks generally represents a proposition, and the activation of the unit represents the degree of truth or likelihood of the proposition. Links represent interactions between propositions; propositions can either activate or inhibit each other. The networks are run by allowing them to settle to a stable state. Some of the more formal work with localist networks treats links as constraints between propositions, and analyzes the settling of the network as a search for an energy minimum.

One of the main attractions of this type of model is its style of computation, in which multiple and diverse constraints are satisfied in parallel. This contrasts with the generally sequential style of computation in rule-based models. In rule-based models much time can be wasted searching entire classes of answers that will be rejected en-masse by constraints applied later in processing.

The representations are generally local in the sense that units represent propositions in a one-to-one fashion. However, as pointed out in the discussion of the local versus distributed issue (Section 1.4.3), sometimes local representations can be considered to be distributed at a higher level, at which the units are the features of objects. Derthick [1990] emphasizes this point, and his model is sensitive to similarities induced by sharing representational units.

In all of these networks, relevant propositions are made explicit by devoting units to represent them. It is this enumeration of propositions which makes possible the simple style of computation. Everything that needs to be considered, including all possible interactions, is laid out explicitly. Connections usually encode pairwise interactions between atomic entities, so interactions among more than two units must be expressed as several pairwise interactions, which requires more units. The downside of this simple and explicit style of representation is that networks can become very large when there are many possible propositions. Holyoak and Thagard's [1989] ACME, which performs analogical matching, is an example of this; the propositions are hypothesis about matches between structure elements, and there are many possible matches when the structures contain many elements.

In most of the networks described here the representations and the interactions are constructed by hand or by a computer program from a priori knowledge, rather than being learned from examples. Shastri's system is an exception to this – its connection strengths are derived from frequency observations. There are other localist systems which learn connection strengths, e.g., Neal's [1992] and Pearl's [1988] belief networks. Furthermore, general schemes such as the Boltzmann machine learning algorithm [Ackley, Hinton and Sejnowski 1985] can be used to learn connection strengths in a localist network.

I review six models here: three for parsing natural language, two for doing simple inference, and one for mapping analogies. Natural language processing is a popular task because it requires complex temporary data structures, and more processing power than provided by a finite state machine. Explicit representation is a common theme in these models – and it results in large networks when many combinatorial interactions must each be represented with separate nodes.

2.1.1 Interpreting and parsing language by constraint satisfaction

Waltz and Pollack: "Massively Parallel Parsing"

Waltz and Pollack [1985] devise a "Massively Parallel Parsing" model of natural language interpretation, which integrates constraints from diverse sources into a homogenous model. The constraints range from syntactic, i.e., word order, phrase structure, etc, through semantic, e.g., selectional restrictions on case-roles of verbs, and contextual, to pragmatic, e.g., typical situations. Figure 2.1 shows a network which integrates all the constraints for parsing and interpreting the sentence "John shot some bucks." One goal of their research is to demonstrate a system that could apply all of these constraints in parallel during the computation. They consider this style of processing more efficient and more psychologically plausible than rule-based systems which perform syntactic processing first, followed by semantic analysis, etc.

Their model uses different units to represent different syntactic interpretations of words or phrases and different senses of words. Links among the units are either inhibitory, for mutually exclusive interpretations, or excitatory, for consistent interpretations. When run, the network settled to a stable interpretations after fifty or so cycles.

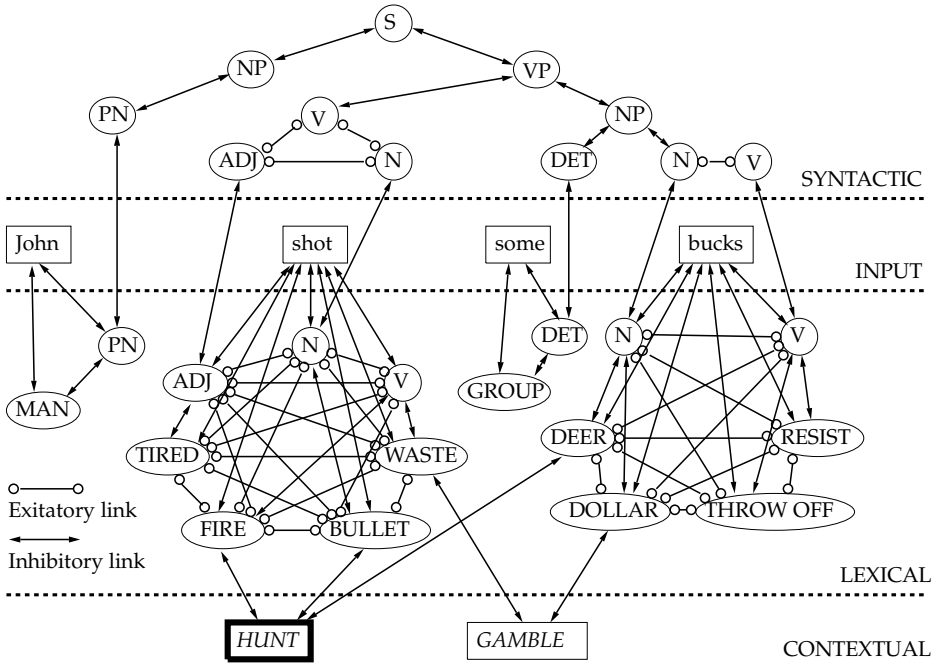


Figure 2.1: Waltz and Pollack's network for parsing and interpreting the sentence "John shot some bucks". (Adapted from Waltz and Pollack [1985].)

The system constructs a new network for each sentence presented to it. It uses a chart parser to find all possible syntactic parses, and translates these into a network form. Units standing for propositions corresponding to different parses are connected by inhibitory links, and units standing for propositions corresponding to the same parse are connected by excitatory links. The example shown has only one full parse, but nodes are constructed for the alternative parts-of-speech of the words. The system adds further nodes and links to the network to represent the semantic and pragmatic interpretations and constraints.

One serious limitation of this model as a technique for language understanding is that a conventional symbolic computer program must construct a network which embodies all possible syntactic and semantic interpretations of the sentence. It seems that most of the work of parsing and interpreting is being done by this program – the connectionist network is only choosing among pre-computed alternatives. Another limitation is that all possible units and links must be supplied to the system in advance. Waltz and Pollack discuss this issue and speculate that distributed representations with microfeatures might somehow be used to overcome this problem.

Cottrell and Small's [1983] model of word sense disambiguation has many similarities to Waltz and Pollack's model. It differs in that a single network is used to deal with all stimulus, but the part of the model that analyzes syntax is undeveloped. They acknowledge that as the network gets larger, i.e., as more words, senses, or syntactical alternatives are added, it gets difficult to control – the network can settle into states which do not represent

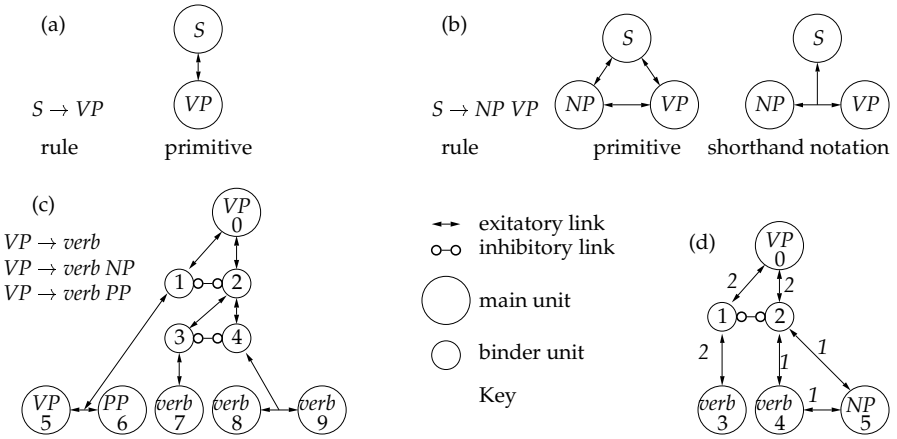


Figure 2.2: Components of Selman and Hirst's network for parsing sentences from a simple grammar. (a) and (b): two examples of grammar rules and the connectionist primitives which implement them. (c): How binder units (units 1, 2, 3, and 4) are used to implement three rules for *VP*. (d) The weights on excitatory links in part of a typical parsing network. (Adapted from Selman and Hirst [1985].)

any sensible interpretation of the input.

Selman and Hirst: parallel parsing network

Selman and Hirst [1985] show how a context-free grammar can be encoded in a connectionist network, so that parsing is accomplished by letting the network settle to an energy minimum. Unlike Waltz and Pollack's network, their network can parse any sentence (up to some pre-specified maximum length). However, it does not incorporate any semantic or pragmatic constraints. The advantage of their system over traditional parsers is that rules are applied in parallel, and that top-down and bottom-up processing is integrated. They show how to derive the weights and thresholds from the rules in a principled manner, which guarantees that the system will find the correct parse when run as a Boltzmann machine to thermal equilibrium.

Each grammar rule is translated into a small network, and networks for rules are combined with "binder" units (Figure 2.2a.) Alternative productions (i.e., the right-hand sides or bodies of rules) are linked by inhibitory connections and the head of a rule is linked with excitatory connections to the nodes in the body of the rule. The weights are set according to a simple scheme. The system does not learn from experience, though it is conceivable that the weights on the links could be learned using the Boltzmann machine learning algorithm. Constituents can be shared using binder units, rather than duplicated, which can save many units and links. Figure 2.3 shows the grammar rules and a network for parsing sentences of up to five words.

To parse a sentence, the lexical types of the words are instantiated in the bottom row of the network. The first word goes in input group 1, the second in input group 2, etc. If a word has more than one possible lexical type, then more than one unit in the group can be

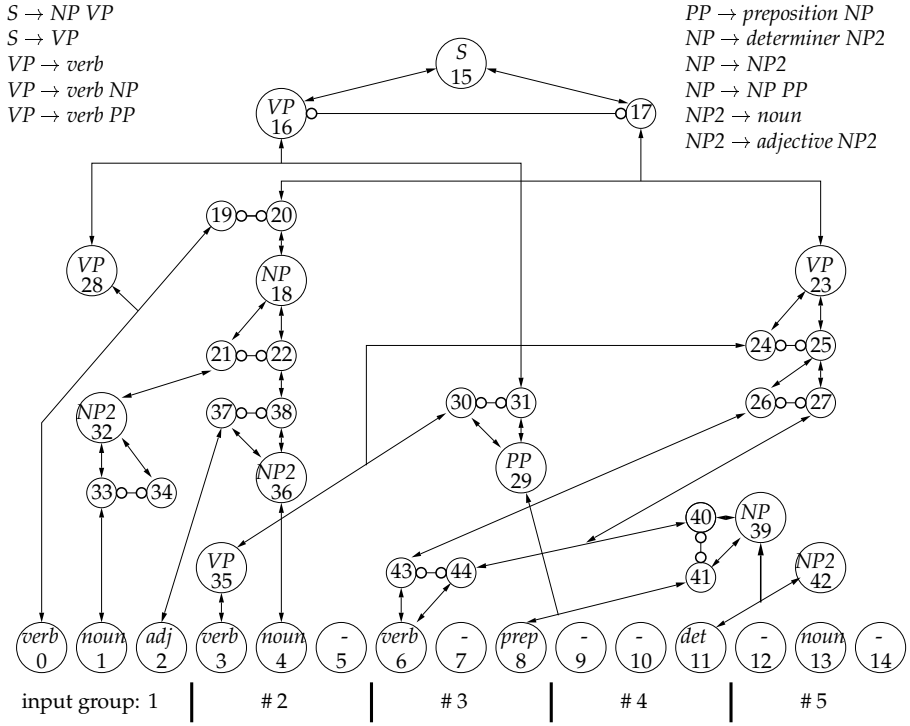


Figure 2.3: Selman and Hirst's simple grammar and the network for parsing sentences generated by it. The network can handle sentences with up to five words. (Adapted from Selman and Hirst [1985].)

turned on. The network is annealed, and when it reaches thermal equilibrium the correct parse can be extracted from the average activations of the units.

Again, this network can use very simple computation because it has a very explicit representation. The nodes in the network can be seen as an enumeration of the ways in which a sentence can be parsed. There is a node for every non-terminal in every position it can appear in the parse tree. For example, Node 32 is the *NP2* that can begin at the first word, Node 36 is the *NP2* that can begin at the second word, and Node 42 is the *NP2* that can begin at the fifth word.

One problem with this system is that there is a hard maximum on the length of sentences that it can parse. Another is that the number of units grows rapidly with the maximum length of sentence to be parsed (n^2), and the time to reach thermal equilibrium grows with the number of units. For unfrustrated systems, the time to reach thermal equilibrium can scale well. However, if there are high energy barriers, Boltzmann machines take a very long time to reach thermal equilibrium.[1] In practice, the system can only be run for a finite

[1]Geman and Geman [1984] show that approach to thermal equilibrium is guaranteed if the temperature at time k, $T(k)$, satisfies $T(k) \geq \frac{c}{\log(1+k)}$, where c is proportional to the maximum height of energy barriers. Since we are interested in the distribution at low temperatures, the settling times required can be very long.

time, which means that the parse found is not guaranteed to the best one.

Charniak and Santos: parsing network

Charniak and Santos [1987] attempt to overcome the fixed length limitation of systems such as Selman and Hirst's by having a window of processing units past which the inputs are slid. The title of their paper gives some hint as to how they view this attempt: "A Connectionist Context-free Parser which is not Context-free but then it is not really Connectionist either".

The network consists of a rectangular array of cells, the bottom row of which contain units for each terminal symbol of the language (the lexical types of words – noun, verb, determiner, etc). Each of the cells in the rows above contains a unit for each non-terminal in the language (sentence, noun-phrase, etc). A sentence is presented to the network by activating the appropriate units in the bottom row of the rectangle – one word per column. After the network settles the parse tree of the input is represented in the activations of units. Each column of units represents a path from a leaf to the root. Links to units above and below and in adjacent columns constrain the system to represent valid parse trees. Other binding units indicate when units in adjacent cells represent the same non-terminal. If the sentence has more words than there are spaces in the rectangle, the input is shifted by one column and the network allowed to resettle. This continues until the whole sentence has been processed. As words slide out of the back of the network, their interpretation (i.e., their path to the root) is presumed stable and is recorded. The total parse is considered to be what is in the system and what has slid out the back. The system is limited in the height of parse trees it can represent, but the parts of the parse tree that get pushed out the top can be treated in the same way as the parts that slide out the back.

As with Selman and Hirst's connectionist parser, the connection strengths in Charniak and Santos's network are not learned, but are derived from the context-free grammar rules. The interesting thing about this network is that it overcomes, to some extent, the fixed sentence length restriction of other connectionist parsers. The memory limitations of the network result in it being able to cope with some types of long sentences better than others. It can correctly parse sentences with right-branching parse trees, because the parse of later segments of the sentence does not depend upon the exact parse of earlier segments. However, it is unable to find the correct parse for centre-embedded sentences because relevant parts of the parse tree slide out the back of the network and are no longer available when they eventually are needed. It is not clear if there is any deep relationship between these limitations and the relative difficulties people have with these classes of sentences.

2.1.2 Simple reasoning

Shastri: semantic networks

Shastri [1988] describes a connectionist network which can efficiently compute a useful but limited class of formal reasoning. The network can perform property inheritance, which is finding the properties of a concept, and recognition, which is finding the concept that best matches a set of properties. Shastri uses "Evidential reasoning" to deal with exceptions, multiple inheritance, and conflicting information in a principled manner. In contrast to other connectionist networks, there are proofs that this network computes the correct answer in a strictly limited time. This comes at the cost of expressiveness and power

– Shastri has picked an extreme point in the tractability-expressiveness tradeoff identified by Levesque and Brachman [1985].

In Shastri's network, nodes represent concepts, properties, and individuals, and links represent relations between them. There is no learning involved – the architecture of the network is derived from statements written in the formal "knowledge level" language of the system. The processing is a form of spreading activation and inference is completed in time proportional to the depth of the conceptual hierarchy. The network can only reason about the properties of a single object at any one time.

Shastri's formulation of the inheritance and recognition problems permits them to be solved in time linear in the depth of the hierarchy (on parallel hardware). Previous formulations of inheritance, e.g., Touretzky's [1986b] inheritance networks, or Etherington and Reiter's [1983] inheritance hierarchies with exceptions, allowed problems that could take exponential (serial) time to solve.

Ajjanagadde and Shastri [1991] describe an enhancement of this network which allows simple rule following. They use temporal synchrony in the phase of unit activations to bind variables from rules with values – a network cycle has a number of phases during which units can be active.

Derthick: μKLONE

Derthick's [1990] μKLONE system for "Mundane reasoning" is another implementation of a formal logic in a parallel constraint system. The language implemented is a subset of Brachman's [1985a] KL-ONE: all constructs except number restrictions, role chains, inverse roles, and structural descriptions are provided.

The type of reasoning the system did is quite limited – hence the name "mundane reasoning". The system finds the most likely set of properties and role fillers for an individual in a particular situation. Although it is based on a logical language, it performs "foolhardy" reasoning in that the assignment of truth values is based on one model of one extension of the rules of the system. This sacrifice of logical soundness is made so that the system can find an answer in a reasonable time, in the belief that some answer, even if sometimes wrong, is better than none at all. The system can deal with inconsistent facts or rules and still find a plausible interpretation.

The system is derived from a specification of frames and relations written in a KL-ONE like language. To process a query about an individual the system builds a network with the individual, its properties, its roles, its role fillers and properties of role fillers. Inheritance is performed by the compiler. The resulting network has a unit representing the conjunction of each role and each filler (even anomalous ones), a unit representing the conjunction of each property and each filler, and units representing the properties of the individual concerned. The links between units are constraints derived from the propositions and rules concerning the concepts in the network. Simulated annealing is used to search for the model with the lowest energy (which has most constraints satisfied).

The representation in μKLONE can be seen as both local and distributed. Simple concepts are represented by single units, but higher level concepts are represented by patterns of the simple concepts. In one example "sailing" is originally the filler of the "job" role for "Ted". However, when the system is told that Ted is a millionaire playboy, and thus does not perform manual labour for a living, it shifts the role that sailing fills to Ted's hobby. This is possible because the job role and the hobby role are similar, both being instances of the "has-interest" role.

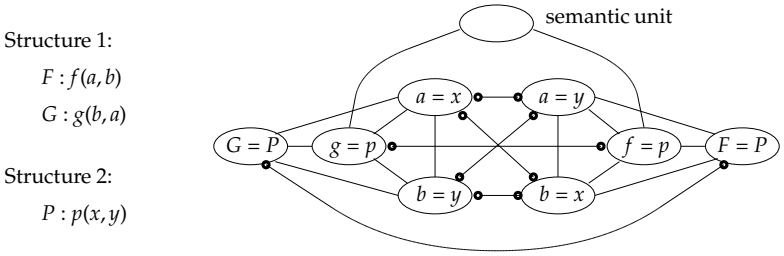


Figure 2.4: The network ACME would construct to find an analogical mapping between the two sets of propositions $\{F : f(a, b), G : g(b, a)\}$ and $\{P : p(x, y)\}$.

Like Shastri's system, μKLONE does not learn – the network is derived from rules written in a formal language. μKLONE also reasons about just one entity, but can represent temporary associations with other roles and fillers. μKLONE is unable to represent complex hierarchical structure, because it only represents immediate relationships to other entities and has no way of representing a chain of relationships.

2.1.3 Analogical mapping

Holyoak and Thagard: analogical mapping by constraint satisfaction

Holyoak and Thagard [1989] describe a localist connectionist model which finds a good analogical mapping between two structured descriptions. The descriptions are in the form of sets of predicates, with nesting allowed. A good mapping is defined by three constraints: isomorphism, semantic similarity, and pragmatic centrality. A mapping defines a one-to-one pairing of the elements from each structure. The isomorphism of a mapping is high when mapped elements are in the same relationships in each structure. The mapping can be incomplete, but more complete mappings are preferred. The semantic similarity of a mapping depends on the similarity of the mapped predicates, and the pragmatic centrality depends on whether the mapping involves predicates which are believed a priori to be important.

The model is embodied in a computer program called ACME (Analogical Constraint Mapping Engine). ACME constructs a connectionist network in which nodes represent competing and cooperating hypothesis about mappings between the elements of the structures. Nodes representing competing mapping hypothesis are connected by inhibitory links, and nodes representing complementary hypothesis are connected by excitatory links. The network is run and allowed to settle to a stable state, and the active nodes represent the mapping found by the network.

Consider the two structures $\{F : f(a, b), G : g(b, a)\}$ and $\{P : p(x, y)\}$. F, G, and P are particular instances of predicates, f, g, and p are predicate names, and a, b, c, d, x, and y are atomic objects. Figure 2.4 shows the network ACME would build to find an analogical mapping between these two structures. It has a node for each possible pairing of elements from the structures, where elements are atomic objects, predicate names and whole predicates. There are inhibitory connections between nodes which represent inconsistent mappings, e.g., $a = x$ and $a = y$. These connections all have the same weight and serve to constrain the mapping to be one-to-one. There are excitatory connections between

nodes which represent mappings of elements of the same predicate, e.g., between $F = P$, $f = p$, $a = y$ and $b = x$. These connections all have the same weight and support isomorphic mappings. ACME builds a "semantic unit" and connects predicate name mapping units to it with an excitatory link whose strength is related to the degree of similarity between the names. If g were similar to p, then the connection between the $g = p$ and the semantic unit would be strong. There is also a "pragmatic unit" which has excitatory links to predicate name mapping units which involve a predicate deemed important (the pragmatic unit is not shown in this figure). This network has two stable states; in one the nodes on the left are active, in the other the nodes on the right are active. These stable states represent P mapping to G, and P mapping to F. Mappings such as $\{G = P, g = p, a = y, b = x\}$ are not isomorphic and will not be stable states of this network. Note that ACME does not build an explicit representation of either structure – it only builds an explicit representation of the mapping between the structures, and the constraints on the mapping.

There is no guarantee that the state ACME settles in will be the best mapping or even a one-to-one mapping, but it appears to almost always find good solutions. ACME works because every possible correspondence between the two structures is represented explicitly by a node, and the compatibility of the correspondences is represented by connections between the nodes. The networks that ACME builds can be large – if the size of the structures is n, the size of the ACME mapping network is $O(n^2)$.

2.2 Distributed connectionist models for simple structure

One of the big problems with local representations is their inefficient use of resources, which can lead to a proliferation of units. Distributed representations offer the potential for more efficient use of resources. In this section I review some models which use distributed representations for objects and encode simple relations among them.

2.2.1 Learning associations between objects

To represent relations in a neural network, we must have representations for both objects and the associations among them. Early work in this area typically uses pre-coded distributed representations for objects. Associations among pairs or triples of these objects are represented in the connection weights, which are found by some sort of learning procedure.

Hinton: implementation of semantic networks

Hinton [1981] describes a distributed connectionist network which implements a content-addressable memory for relations. This is one of the first attempts to represent relational data in a distributed fashion. Instead of representing atomic concepts by single units, Hinton represents concepts by patterns of activity over groups of units, and represents the relations between them by interactions among the groups of units.

The network stores two-place predicates, which are triples consisting of a relation name and two roles. A typical set of triples is:

(John has-father Len)
(Mary has-father Len)
(John has-sister Mary)
(Kate has-father John)

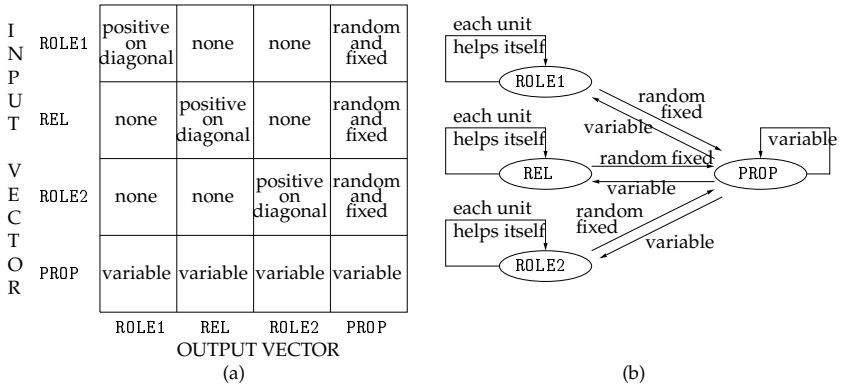


Figure 2.5: Two schematic representations of the connections in Hinton's triple memory. (a) shows the matrix of weights, which connect the units, in block form. Many of the submatrices are null, e.g., the weights from the ROLE1 units to the REL units. (b) shows the groups of units, each arrow represents a submatrix of weights connecting the groups. The matrix of weights from a group to itself (except for the PROP group) is the identity matrix, it is intended to cause the group to retain whatever pattern it is initialized with. Reproduced from Hinton [1981].

The principle task the network performs is associative retrieval (pattern completion). Given any two concepts in a triple, the network finds the matching stored triple. Given the query (John has-sister ?) the system should fill in the blank with 'Mary', and given the query (Kate ? John) the system should fill in the blank with 'has-father'.

The model has a separate group of units for each element of a triple. A concept, such as 'John' or 'has-father', is represented by a pattern of activity over the appropriate group of units. The patterns of activity are pre-coded and are chosen so that similar concepts have similar patterns of activity.

The network architecture is shown in Figure 2.5. In addition to the three groups of units for each element of the triple, there is a fourth group of units labelled PROP (for "proposition"). The PROP units help to implement interactions among the different groups, and are necessary because direct connections between groups are too limited in the types of interactions that they can implement. A triple is retrieved by instantiating two of the groups with the known pattern and then allowing the network to settle to a stable state. A triple is stored in the network by instantiating its concept patterns on the three groups of units and then adjusting the variable weights so that the pattern is stable.

Hinton discusses how this type of model can also implement a form of property inheritance. As mentioned before, similar concepts (or individuals) are represented by similar patterns. Hinton suggests that the representation of a super-type concept should be the set of features that all its subconcepts have in common. Relations that depend on the type of the individual can be implemented by interactions among the microfeatures that are common to individuals of that type. Exceptions can be implemented by interactions among the microfeature patterns that are specific to the exceptional individuals. This is illustrated in Figure 2.6 for three elephants. There are two microfeatures common to all elephants, these code for the 'elephant' type, and the corresponding units in ROLE1 interact

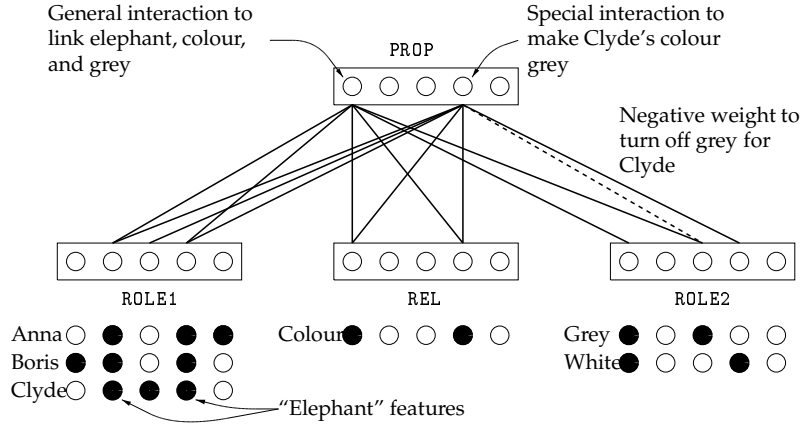


Figure 2.6: Implementation of inheritance and exceptions in Hinton's [1981] triple memory. The solid links indicate positive weights, the dashed link indicates a negative weight, and weights in opposite directions have the same sign. The first unit in the PROP group comes on when an elephant and 'colour' are instantiated in the ROLE1 and REL groups, it causes 'grey' to be instantiated in the ROLE2 group. The fourth unit in the PROP group detects the special case of Clyde's colour being asked for, and suppress the 'grey' pattern, and excites the 'white' pattern. In an actual trained network the interactions in the PROP group would probably be implemented by many units.

with the colour pattern in REL to support the grey pattern in ROLE2. Most elephants, e.g., Boris and Anna, will be thought by the system to be coloured grey. However, Clyde is an exceptional elephant, and the microfeature pattern unique to Clyde interacts with the colour pattern to suppress the grey pattern and support the white pattern, overriding the default inference that Clyde is grey.

There are two types of conceptual structure in this model. The pattern of microfeatures that represents a concept constitutes the "direct content" of the concept. The links to other concepts constitute the "associative content" of the concept. Hinton suggests these two types of conceptual structure correspond to two separate components of human memory sometimes called "integrative" and "elaborative" structure. The first has to do with internal coherence, and the second to do with external relations. The interesting thing about this model is that direct content of the concepts causes the associative content. Furthermore, small changes in the direct content (pattern) will usually lead to small changes in the associative content, which results in the system having some automatic inheritance and generalization abilities.

One limitation of this model is that it can only represent one relation at a time. It cannot represent multiple relations, let alone a structure among a set of relations. Another limitation is that there are a fixed number of roles in the system. Adding more groups of units for more roles is not a good solution because it is wasteful for relations that do not use all the roles, and because it makes it difficult to represent similarity among roles. Hinton suggests that roles could also be given distributed representations and role-filler bindings represented by coarse conjunctive codings. I discuss this approach in Section 2.2.3.

Anderson, Spoehr and Bennet: Arithmetic

Anderson, Spoehr and Bennet [1991] describe a model of learning multiplication tables which uses analogical distributed representations. Their representation of a number has two components: an analogical component in which numbers of similar magnitude have similar representations (using a "sliding bar" method), and a random component. The representation is very wide – 422 units for each number. The random component is necessary to make the representations sufficiently different, but the interesting properties of the model arise from the analogical component of the representation. The model is trained to remember triples of two operands and one result, using a "Brain-state in a box" network [Anderson et al. 1977] (a type of auto-associative memory).

As the title indicates ("A Study in Numerical Perversity: Teaching Arithmetic to a Neural Network"), this model is not intended to be a good way of having a machine remember multiplication tables. Rather, it is a demonstration of how a simple connectionist model that uses analogical distributed representations can model human performance. The overall error patterns and the relative response times for false products (people are quicker to identify $5 * 3 = 42$ as incorrect than $8 * 7 = 63$) are similar to people's performance on the task of learning multiplication tables. These performance characteristics are a direct consequence of the analogical structure of the representations of the numbers.

Rumelhart and McClelland: learning past-tenses of verbs

Rumelhart and McClelland [1986a] describe a distributed connectionist model which learns to produce the past-tense forms of regular and irregular verbs. They represent words as phonemic strings in a distributed fashion over about 500 units. Each word is represented by a set of phonemic triples, which in turn are represented by sets of finer-grained conjunctions called "Wickelfeatures". The set of Wickelfeatures for a word is stored in the distributed memory. For example, the word 'sit' would be represented by three triples: '#si', 'sit', and 'it#', where '#' is the start and end symbol. This representation makes local order explicit, but leaves global order implicit – it is a puzzle to see how the triples of a long word could fit together. It is possible that different words could have the same representation, but this problem does not arise with the words used in the model. The network takes the representation for the root from of verb ('sit') as its input and is meant to produce the representation for the past-tense form ('sat') on its output. Rumelhart and McClelland trained a noisy perceptron-style map to do the mapping from input to output.

Although the network eventually learns to produce the correct past-tense forms for both regular and irregular verbs, the interesting thing about it is pattern of errors it makes during learning. Early on in the learning process it produces the correct past-tense forms for irregular verbs (e.g., 'sat' for 'sit'), but as it learns the regular verbs it begins to produce incorrect overgeneralized past-tense forms for irregular verbs (e.g., 'sitted'). In the final stage, it learns the correct forms for both regular and irregular verbs. The fully trained network is also able to generalize; it can produce the correct past-tense form for some words which were not in the training set. Although this model has received extensive criticism as a psychological model of acquisition of verb tense (e.g., from Pinker and Prince [1987], who point out that the gradated training set might have something to do with the time-course of errors), it is still interesting as a demonstration of how rules and exceptions can be learned using a single homogeneous network.

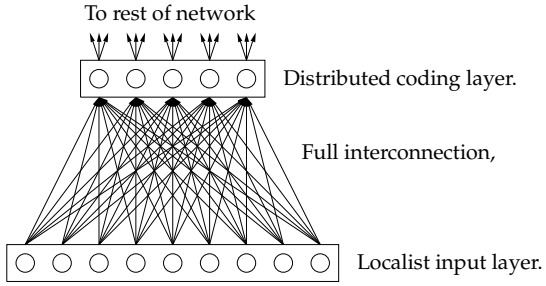


Figure 2.7: An extra intermediate layer in a feedforward network can convert from localist to distributed representations Only one unit in the input layer is active at any time. The links from the active input unit determine the activations on the units in the distributed coding layer.

2.2.2 Learning distributed representations for objects

The models we have seen so far have used hand-coded feature-based distributed representations. For reasons mentioned in Section 1.4.2, it is better if a system can develop its own distributed codes. Hinton [1986] shows how this can be done in a feedforward network by using localist input layers and extra intermediate layers. The method involves connecting each localist input layer to its own distributed coding layer, as shown in Figure 2.7. Only one unit in the input layer is ever active, so its weights determine the activations on the coding layer. These weights can be trained along with the other weights in the network. This scheme does not significantly increase the amount of computation, because only the weights from the active unit to the coding layer need to be considered in both the forward and backward computations.

Usually, the distributed coding layer has far fewer units than the localist input layer (this is often called a "bottleneck"). This is done for two reasons, the first being to keep the number of parameters down, so that not too many examples are required to train the network. The second reason is to force the network to develop useful distributed codes. If the code layer has fewer units than the input layer, then the network must develop some sort of distributed representation on the coding layer, just to be able to represent inputs with different codes. The hope is that these distributed codes will be based on commonalities the network discovers while learning to perform the task. Indeed, this is what appears to happen in the networks that use this technique. Recently, Zemel and Hinton [1994] have shown that it is possible to use a wide distributed coding layer and still learn useful codes, by imposing various constraints on the codes.

This technique can also be used for output representations. However, doing this can significantly increase the amount of computation, because it is not possible to ignore any of the connections from the distributed output code to the localist output code, except by making some risky approximations.

Hinton: "Family trees" network

Hinton [1986] demonstrates that a backpropagation network can learn good distributed representations in the course of learning to perform a task. The network he describes learns

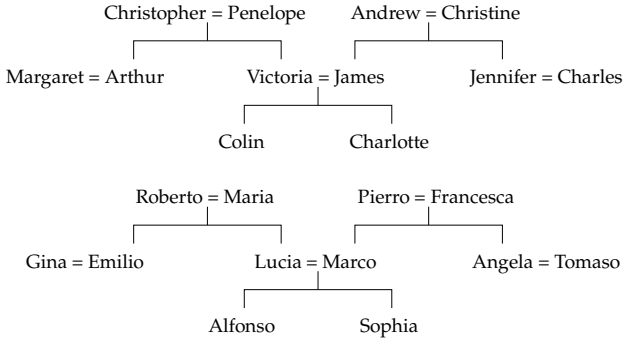


Figure 2.8: The family trees stored in Hinton's [1986] "Family trees" network. The two trees, for English (top) and Italian (bottom) people are isomorphic.

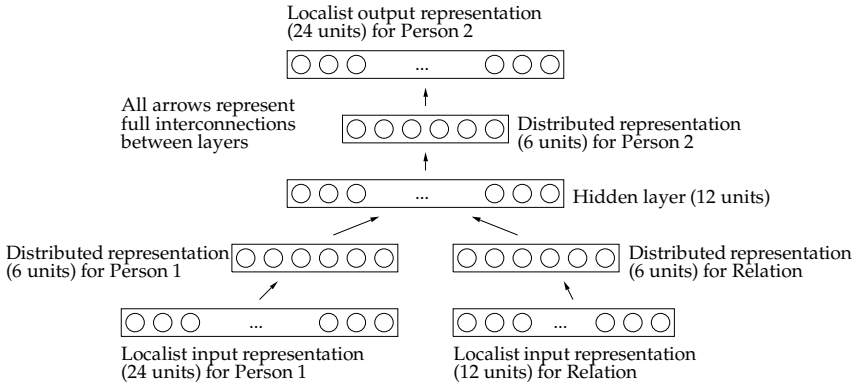


Figure 2.9: Architecture of Hinton's [1986] network for storing family trees as a set of relationships.

the relations embodied in the two isomorphic family trees shown in Figure 2.8. The form of the relations is (Person1 Relation Person2), e.g.:

Sophia	has-aunt	Angela
Charlotte	has-aunt	Jennifer
Charlotte	has-father	James

The network is a feedforward architecture which, given a person X and a relationship R, completes the relation by outputting the person who is in relation R to person X. The architecture of the network is shown in Figure 2.9. The network uses bottleneck local-to-distributed mappings on the inputs and outputs. There are 24 individuals, and six units in the encoding units, so the network must learn a relatively efficient distributed encoding.

Hinton trained the network using the backpropagation with weight decay on 100 of the 104 instances of relations in the two family trees. The trained network successfully processes the remaining four relations, indicating that the learned representations and

mappings can support generalization. Presumably, the isomorphism of the two trees makes the knowledge about mappings transferable across nationalities. Upon examination, some of the units in the input encodings for people were found to stand for identifiable features of the individuals, e.g., nationality and generation. Sex is not encoded in any unit – it is actually not useful in determining the correct output. The network does not "know" that nationality and generation are meaningful features, it is only told things like "when unit 19 in input block 1 and unit 3 in input block 2 are on, turn on unit 13 in the output block". It gives some objects similar representations merely because this helps to solve the mapping problem. It turns out that these similar representations can be interpreted in terms of common features.

Although this network stores relations, it is not an auto-associative memory like Hinton's triple memory (Section 2.2.1). This network can only produce Person 2 given Person 1 and the relationship – it cannot produce Person 1 or the relationship given the other two elements of the relation. The reason for this limitation is the difficulty of incorporating local-to-distributed mappings in recurrent auto-associative memories.

Other networks which learn distributed representations

Harris [1989] uses this same technique in a system that learns to categorize the meanings, according to Cognitive Linguistics, of the preposition 'over' in sentences. The network learns distributed representations for words which code for properties relevant to how 'over' should be interpreted. The learned representations also exhibit some of the properties of the "radial categories" of Cognitive Linguistics.

When a network has a number of input blocks, weight sharing can be used to force all distributed input groups to use the same distributed representation. This has several advantages – objects have the same distributed representation no matter where they appear in the input, the distributed representations combine information from many sources, and the network has less parameters. Lecun, Boser, Denker, Henderson, Howard, Hubbard, and Jackel [1990] implement a spatial version of this technique to learn translation-invariant spatial features in their model of handwritten digit recognition.

Miikkulainen and Dyer [1989] extend this local-to-distributed mapping technique to force the distributed output representation to be the same as the distributed input representation. In Hinton's "Family trees" network 'Charlotte' can have different representations on the distributed output units and the distributed input units. Miikkulainen and Dyer dispense with the localist output layer, and instead give the network a target pattern on the distributed output layer. This target pattern is the distributed input representation of the target object currently used by the network. They use this technique in a network which processes sentences, and the network learns representations for many varied objects. The resulting representations have many analogical properties. The advantage of using the same representation on input and outputs is that all the information in the training set is used to construct a single representation for each object. However, the targets change as the distributed input representations change, which can make learning slower (a "moving targets" problem). A potentially serious problem with this technique is that the network can give all objects the same representation, which results in zero error. This points to a deficiency in the error function – it lacks a discriminative term. Miikkulainen [1993] reports that networks rarely give all objects the same representation,[2] and suggests that

[2]This is probably because Miikkulainen does not use the correct derivatives for training the network – he

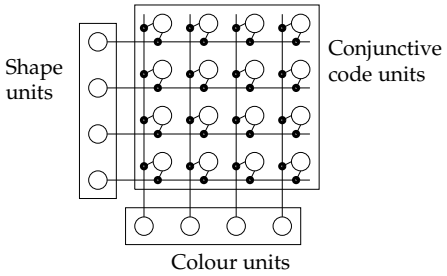


Figure 2.10: A network which implements a conjunctive code for colour and shape. A unit in the conjunctive code is active if both of the shape and colour units it is connected to are active.

this problem could be avoided entirely by clamping at least two of the representations to different values.

2.2.3 Conjunctive coding

Conjunctive coding is a way of binding together the representations of two objects. A conjunctive code has a unit for every possible conjunction of the units used for representing the objects. If the representation for one object has N units, and the representation for another has M units, then the conjunctive code will have $N \times M$ units. Conjunctive coding can be used whenever it is necessary to bind several representations together: for binding roles and fillers, for binding variables and values, for binding the positions and identities of objects, and for binding multiple features of objects together.

Suppose we want to bind colours with shapes, and are using the following four bit distributed representations: 'red' ($\mathbf{r} = [0011]^T$), 'blue' ($\mathbf{b} = [1001]^T$), 'square' ($\mathbf{s} = [1010]^T$), and 'circle' ($\mathbf{c} = [0110]^T$). The conjunctive code for this will have sixteen units. Figure 2.10 shows a network implementation of a conjunctive code – a conjunctive unit is active if both the units it is connected to are active. The conjunctive codes for 'red-square' and 'blue-circle' are shown in Figure 2.11 (the network is not shown). Conjunctive codes can be superimposed without the appearance of illusory conjunctions such as 'blue-square', provided that not too many bindings are superimposed. In the terms of linear algebra, a conjunctive code is a (thresholded) superposition of vector outer products:

$$B = f(\mathbf{r}^T\mathbf{s} + \mathbf{b}^T\mathbf{c}),$$

where f is the superposition threshold-function.

To find out what shape is bound to a certain colour, we treat the colour units as input units and the shape units as output units. If a conjunctive unit is active, and the colour it is connected to is active, then it sends activation to its shape unit. The shape units sum the activation they receive, and output 0 or 1 according to whether their total activation is below or above a suitable threshold.[3] In the terms of linear algebra, this is equivalent to

omits the term which would provide a strong push for all the representations to take on the same value.

[3]With zero-one representations of objects, the appropriate value for the threshold depends on the number of ones in the representations.

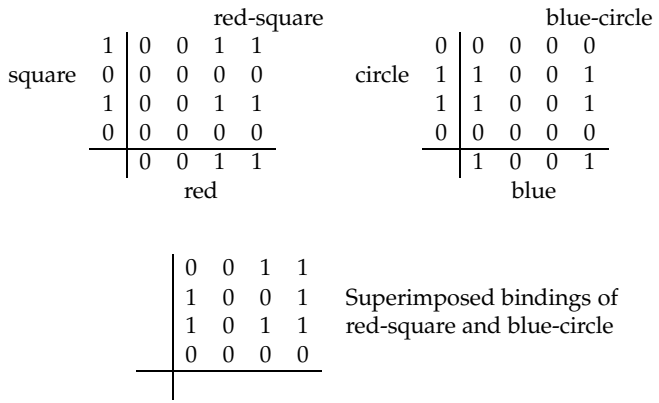


Figure 2.11: The conjunctive codes for the 'red-square' and 'blue-circle' bindings are the outer products of the representations of 'red' and 'square', and 'blue' and 'circle'. These bindings can be superimposed without confusing which colour is bound with which shape.

multiplying the colour vector by the binding matrix, and thresholding the result:

$$f(B\mathbf{r}) = \mathbf{s} \quad \text{and} \quad f(B\mathbf{b}) = \mathbf{c},$$

where f is the decoding threshold-function. This decoding process works in the presence of noise, and with superimposed codes. It can also can be inverted to find the colour given the shape. Figure 2.12 shows the decoding of the superimposed bindings: the procedure correctly discovers that 'red' was bound to 'square' and 'blue' to 'circle', despite the overlap in the representations.

Hinton [1981] discusses how distributed conjunctive coding for roles/filler bindings can solve two potential problems with systems that use blocks of units to represent the fillers of different roles, such as the memory for triples (Section 2.2.1). One problem is the number of role blocks required. A system which could represent a variety of predicates with role blocks would need a large number of them for all the different roles, and could become unmanageably large.[4] Additionally, such a system would be representationally inefficient, as only a small fraction of the blocks would be in use at any one time. Conjunctive coding with distributed representations of roles can make better use of representational resources – one conjunctive code block can represent the binding of a filler with one of many roles. The other problem is that having different blocks for different roles makes it difficult to represent similarity between roles. Using distributed representations for roles allows the similarity structure of roles to be reflected by their representations. These similarity relationships are preserved by conjunctive coding – the bindings of a filler with similar roles are similar.

Conjunctive codes work with both distributed and local representations, but can grow very large, especially when higher-order conjunctions are used. This is more of a problem for local representations than for distributed representations, because local representations

[4]Actually, a system which uses different blocks of units for different roles can be regarded as using a conjunctive code with an inefficient local representation for roles. The active unit in the role representation indicates which row (or column) of the binding matrix should be used to store the filler.

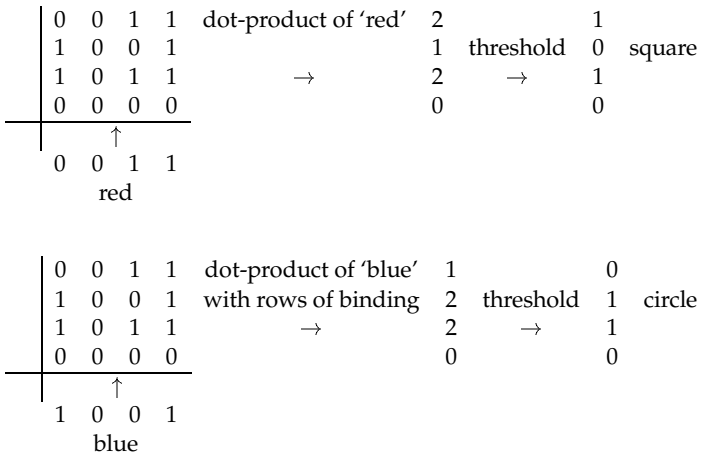


Figure 2.12: Decoding the superimposed bindings of 'red-square' and 'blue-circle', to discover which shape is bound to 'red' and which shape is bound to 'blue'. The first operation is the dot-product of the colour representation with the each row of the superimposed bindings, the second is comparison with a threshold.

require a larger number of units to represent a given number of objects. Conjunctive codes also work with continuous-valued distributed representations: the activation of a unit in the conjunctive code is the product of the activations of the units it is connected to. The use of conjunctive codes generally imposes some constraints on the distributed representations for objects, e.g., that they should have approximately uniform density.

Outer-product conjunctive codes are equivalent to matrix-style hetero-associative memories, such as the Willshaw net [Willshaw 1981b], and like them are able to exploit redundancy in the conjunctive code to compensate for noisy or incomplete input patterns. The encoding and decoding operations of matrix memories can be implemented in several different ways, giving rise to a variety of ways of using conjunctive codes. For example, instead of superimposing bindings in a binary-OR fashion (as in Figure 2.11), we can superimpose bindings by ordinary summation. The best choice of operations for a conjunctive code depends upon the properties of the distributed representation of objects (such things as its density, and whether it is binary or continuous).

Outer product codes are the most straightforward conjunctive codes for distributed representations. More complex conjunctive codes can be constructed by involving more than one unit from the object representations in the conjunctions, as Touretzky and Geva do in their DUCS system (described in the next section). This makes it possible to increase redundancy in the conjunctive code, since more than $N \times M$ higher-order conjunctions are available. Increased redundancy gives the system greater ability to correct errors and complete patterns.

Conjunctive coding can also be extended to bind more than two objects together. Smolensky [1990] describes a framework for this, based on tensor product algebra. A tensor product can be viewed as an outer product operation generalized to any number of

vectors. I review tensor products in Section 2.4.3. Tensor products provide a general framework for understanding conjunctive codes – all conjunctive codes, including the complex code Touretzky and Geva use in DUCS, can be viewed as tensor products or subsets of tensor products.

Touretzky and Geva: DUCS

Touretzky and Geva [1987] use conjunctive coding to store simple frame-like structures in a distributed memory. Their system, called DUCS, is able to retrieve a role-filler pair given an approximation of the role, which can be interpreted as generalization or error correction. This ability is a consequence of the use of distributed representations and redundant conjunctive coding.

A frame consists of a number of role-filler pairs. For example, the frame describing "Fred the cockatoo" is:

NAME:	FRED
BODY-COLOUR:	PALE-PINK
BEAK:	GREY-HOOKED-THING
CREST:	ORANGE-FEATHERED-THING
HABITAT:	JUNGLE
DIET:	SEEDS-AND-FRUIT

DUCS can store several role-filler pairs in a frame, and several frames in its memory. DUCS is pattern-completing and error-correcting at the role level and at the frame level. It can retrieve a frame given a partial or corrupted version of the frame, and can retrieve a role-filler pair from a frame given a corrupted or similar version of a role in the frame.

The architecture of DUCS is shown in Figure 2.13. Roles and fillers have binary distributed representations 20 units wide. The binding of roles and fillers is a highly redundant (and rather complex) set of 1280 conjunctions of role and filler units. Each conjunction can involve the activation of a unit, or its logical complement, and the set of conjunctions is designed so that exactly 40 of the 1280 units are active. A "selector" block holds a single role-filler binding and has extra internal units that exploit the redundancy of the conjunctive code to error correct roles and fillers during frame decoding. A frame is a superposition of role-filler bindings, and is stored in the frame buffer over 1280 units. The number of active units in the frame buffer will be approximately the number of roles in the frame times 40. The frame memory is an auto-associative Willshaw net [Willshaw 1981b], which is a pattern-completing and error-correcting memory. It can store a number of frames, and can retrieve one given a partial or corrupted version. The frame memory has $1,638,400$ links.

The representations on the role and filler units are chosen to reflect the similarity of the concepts they represent. This allows the system to retrieve a filler from a binding given an approximation of the name of the role. For example, if the representation for "nose" is similar to that for "beak", then if we request the filler of the "nose" of Fred role the system will change the requested role to "beak" and say the filler is a "grey hooked thing".

Touretzky and Geva discuss how recursive structure (in which a frame can be a filler in another frame) could be represented in DUCS. This is difficult because the representation of a frame is 1280 units, whereas the representation of a filler is 20 units. They suggest that the first 20 units of the 1280 could serve as a reduced description for the frame, and could be used as a filler in other frames. However, this is unlikely to be an adequate reduced

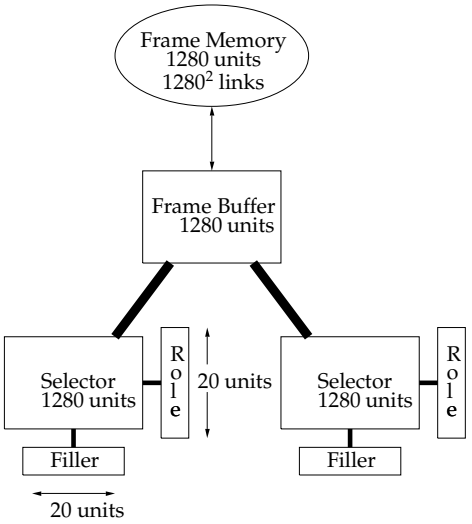


Figure 2.13: DUCS architecture. (Adapted from Touretzky and Geva [1987].)

representation. The code for a frame is sparse, so in any 20 units of a frame there will only be a few active units. If there are a reasonably large number of frames, the chances are that some will have the same reduced description.

2.3 Connectionist models which learn to process language

The prospect of processing natural language has been a prime motivation for the interest among connectionists in the representation of complex structure. Typical language processing tasks are: deciding whether a string is grammatical, predicting the next symbol in a string, assigning case-roles to words in a string (i.e., deciding which is the subject, object, etc), translation and pronunciation. On top of the problems with representing and manipulating complex temporary structures in neural nets, several related characteristics of language add to the difficulty. The input consists of a variable number of objects, typically in sequential form. Furthermore, the appropriate output at any particular time can depend upon input received an arbitrary number of timesteps in the past. A network which can only deal with sequences up to some fixed length will be of limited use, and any learning it does is unlikely to reveal much of interest about how language can be processed. To process language in a satisfactory manner, it is necessary to use some sort of network which is capable of sequential processing, can retain complex state information, and has context-dependent control over what state information is retained.

Inspired by the ability of feedforward networks to develop useful representations in the course of learning to perform a task, some researchers tried this approach with recurrent networks on language processing tasks. Recurrent networks are a modification of feedforward networks in which the activations of some units are considered to be the state of the machine and are recycled as inputs to the network at the next timestep. Such a network can process a sequence of inputs and maintain an internal state. However, before I discuss

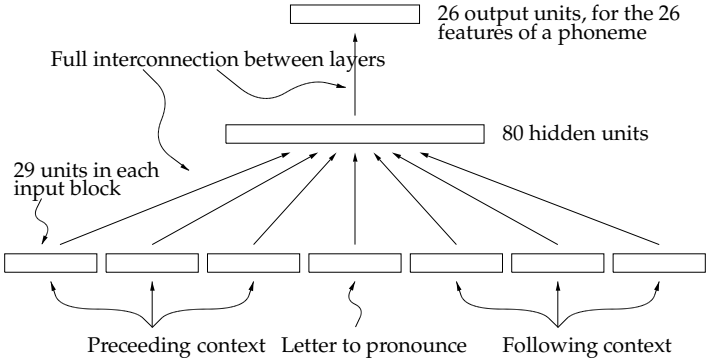


Figure 2.14: Architecture of Sejnowski and Rosenberg's [1986] NETtalk system.

recurrent networks I want to review NETtalk, which is a feedforward network that learns to pronounce English text. NETtalk serves to show that neural networks can learn some regularities and exceptions in a language task (English pronunciation), and clarifies the limitations of non-recurrent networks.

2.3.1 Non-recurrent networks

Sejnowski and Rosenberg: NETtalk

Sejnowski and Rosenberg's [1986] NETtalk system, which learns to pronounce English text, is a feedforward network which looks at a seven-character window of text. Its task is to output the phoneme corresponding to the central character in the input window. The architecture of the NETtalk system is illustrated in Figure 2.14. Each input buffer represents one letter (or punctuation mark) in a local fashion, and the output is the set of features for the phoneme for the central letter. For cases where more than one letter generates a single phoneme, the output phoneme can be 'null' for all but one of the letters.

The most obvious limitation of this windowing technique is that the "memory" it provides is strictly limited to inputs that occurred within a fixed number of timesteps. The network has no internal state. Once an input passes out of the window it can have no effect on the output. While having access to the three previous input symbols might be adequate, in most cases, for pronunciation, it is inadequate for more complex language processing tasks. Merely increasing the window size would be inelegant, inefficient, and impractical – the window would need to be quite large to be large enough most of the time, there would be an impracticably large number of parameters, and there would always be otherwise reasonable inputs which were too large.

Another limitation of this windowing technique is that it is difficult to design the system so that it can recognize and exploit invariances in the input. For example, words often mean the same thing independent of which buffer window they appear in, and it is wasteful to duplicate the machinery that recognizes words. A possible solution is to share the weights for local to distributed maps on input buffers. However, the problem is more difficult to solve at higher levels, e.g., in the interpretation of the meanings of short sequences of words.

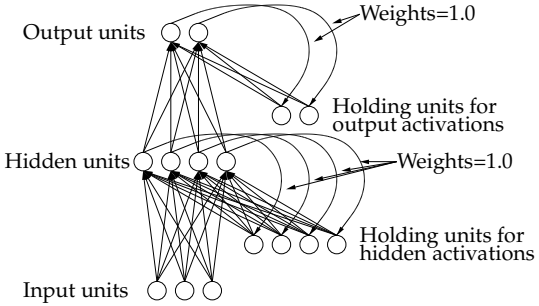


Figure 2.15: A recurrent network, as described by Rumelhart, Hinton and Williams [1986]. The backward connections serve to copy the activations to holding units, for use at the next timestep.

2.3.2 Recurrent networks

Rumelhart, Hinton and Williams [1986] describe how recurrent networks can be trained using a modification of backpropagation algorithm for feedforward networks. In a recurrent network activations on certain units are fed back into the network at the next timestep. Figure 2.15 shows the architecture of one such a network. A complete forward propagation, from the input to the output units is one timestep. The backward connections copy activations to special "holding" units, which preserve the activation for use at the next timestep.[5] In the first timestep all the holding unit activations are set to some fixed value. In subsequent timesteps they assume the activations at the previous timestep of units they are connected to. The activations of any unit in the network can be recycled into the net at the next timestep by including a holding unit for it. The network can have different input values and output targets at each timestep, and inputs and targets can be left undefined at some timesteps. Rumelhart, Hinton and Williams describe how to calculate error gradients for such networks, using an "unfolding in time" procedure. This involves running the network backwards for the same number of timesteps as it was run forwards.

Rumelhart, Hinton and Williams trained a network like the one in Figure 2.15 to complete sequences. They used 25 different sequences six symbols long, e.g., 'AB1223', and 'DE2113'. The sequences have the regularity that the third and fourth symbol are determined by the first, and the fifth and sixth by the second. The first two symbols are presented to the network as input at the first two timesteps. The remaining four symbols, which are completely determined by the first two, are the output targets at the subsequent four timesteps. After training on 20 of the sequences the network could correctly generate those and could generate completions for the sequences it had not seen. In one training run the network generated completely correct completions for the unseen sequences, and in another it made just one mistake. This generalization ability indicates that the network is able to extract the regularity in the training set.

[5] Rumelhart, Hinton and Williams use different classes of links rather than holding units, but the results are the same.

Elman: Learning to remember

Elman [1988; 1990; 1991] performed several experiments with recurrent networks, in order to investigate how well they can process language. The results show that simple recurrent connectionist networks can build representations that stored relevant information from previous input, that these representations seem to possess internal structure, and that this structure can be used by the network to account for structure in the input. However, these results must be taken with some qualifications: the learning is slow, the domains tested on are simple and small, the representations are difficult to interpret, and the scaling and generalization properties are uncertain.

Elman uses recurrent networks in which the activations of the hidden layer are recycled to the next timestep.[6] Figure 2.16a shows a typical network. He calls the holding units the "context" units, because these units hold the memory of previous inputs – they provide the context for the net to work in. The task of the network at the t-th timestep is to predict the $(t+1)$-th symbol in the sequence, given the t-th symbol as input and the context computed so far by the network. In the course of making this prediction new activations are computed on the hidden units. These are used as the context for the following timestep, when the actual $(t+1)$-th symbol is presented on the inputs and the network has to predict the $(t+2)$-th symbol. There are nearly always several symbols which can occur at a given point in the sequence. The network can minimize its error by predicting all possible items, but with lower values ("confidences"). In order to make optimal predictions (and thus minimize the error) the network must discover all the constraints, syntactic and otherwise, on the language it is presented with. It must also learn to preserve relevant information about previous inputs in its context layer, and learn to ignore or discard irrelevant information.

Elman does not compute exact error gradients in the networks. Rather than running the net backwards through the training sequence to correctly compute errors, Elman truncates the backpropagation of error signals at the context layer. Figure 2.16b shows an unfolded network and the forward and backward propagation paths. Truncating the error signals makes the gradient computation much simpler, as it is no longer necessary to store a history of activations at each unit and run backwards through the sequence. The backpropagation of errors can be performed at each timestep, resulting in an algorithm which is local in time. However, the gradients are not correct, since there is no backpropagation of error signals from an output to an input which occurred on preceding timesteps. This means that if an input is only important several timesteps in the future, the network will not adjust its weights to preserve information about that input. If by chance the information is preserved, the network can use it, but the calculated gradients only force the network to transfer information about the input into the hidden layer if that information is useful for the current output.[7]

Elman's simplest experiment [1990] is with a recurrent network that learned the sequential XOR task. In this task a string of bits is presented to the network, and the network must predict the next bit. Every third bit is the XOR of the previous two random bits, and the network should learn to predict its value. For example, in the string "01$\underline{1}$11$\underline{0}$10$\underline{1}$…" the

[6] Jordan [1986] used recurrent networks in which output activations are recycled. However, this type of net is not very suitable for processing language strings because language tasks usually require the retention of state information that is independent of the outputs.

[7] Williams and Zipser [1989] describe an algorithm for correctly computing error gradients which does not require going backwards through the sequence. However, the algorithm is non-local, and expensive both in time and space.

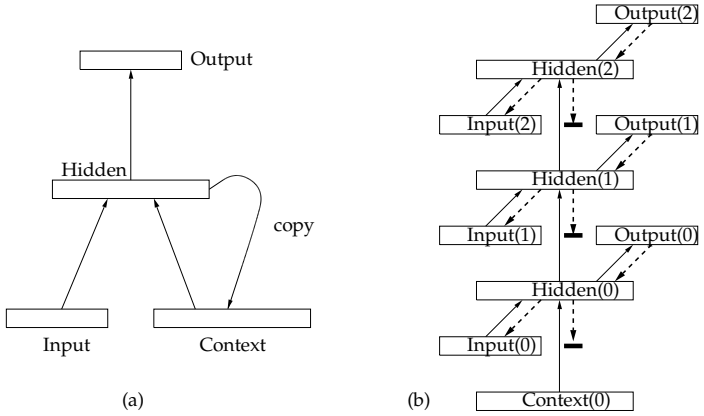


Figure 2.16: (a) The type of recurrent net used by Elman [1990]. (b) The unfolded network showing the patterns of forward and backwards propagations. The numbers in parentheses are the timesteps. Errors are not backpropagated from the hidden units at time $t+1$ to the hidden units at time t.

underlined numbers are predictable. For unpredictable bits, the total error over the training set is minimized when the network predicts both with lower confidence. The network learned this task after a moderately long training period: a string of approximately 3000 bits. This experiment demonstrates that this type of network can learn to remember.

Another experiment involves learning to predict the next word in two- and three-word sentences generated by a simple template grammar with a vocabulary of 29 words. The grammar incorporates some semantic constraints, e.g., that the subject of "eat" must be an animate noun. Elman generated 10,000 sentences from the grammar and concatenated them with no begin or end markers to form a continuous sequence of 27,354 words. The words are represented in a local fashion on the input and output units. The network has one hidden layer with 150 units. It learned to make good predictions in six passes over the data. Elman attempted to analyze the representations by computing the average hidden unit representation in response to a particular word in all contexts. He performed a hierarchical cluster analysis on these average representations and found that they cluster, in a hierarchical fashion into verbs, nouns, animates, inanimates, animals, humans, etc. Elman interprets this as showing that the network, as a result of learning to predict, had discovered the semantic structure of the vocabulary. However, as Elman acknowledges, the observed clusters may be an artifact of the averaging and not due to the learning. The network, whether trained or not, constructs contexts for individual word occurrences. The averaging procedure then constructs average contexts for each word. These would be expected to cluster in way the Elman observed, even without any learning, since similar words have similar contexts. Elman (private communication) reports that this is in fact what happens, although the clusters extracted from an untrained network were less clean than those from the trained network. To test what the network has learned, it would be more appropriate to do a cluster analysis on the outgoing weights from the input layer, as these form a local-to-distributed mapping, and any systematic similarities in these weights must be a result of the learning.

S → NP VP "."
NP → PropN | N | N RC
VP → V (NP)
RC → *who* NP VP | *who* VP (NP)
N → *boy* | *girl* | *cat* | *dog* | *boys* | *girls* | *cats* | *dogs*
PropN → *John* | *Mary*
V → *chase* | *feed* | *see* | *hear* | *walk* | *live* | *chases* | *feeds* | *sees* | *hears* | *walks* | *lives*

Additional restrictions:

- number agreement between N and V within clause, and (where appropriate) between head N and subordinate V
- verb arguments:

  chase, feed → require a direct object
  see, hear → optionally allow a direct object
  walk, live → preclude a direct object
  (observed also for head/verb relations in relative clauses)

Figure 2.17: Elman's grammar which allows embedded clauses. (Adapted from Elman [1991].)

Elman's [1991] most complex experiment uses sentences generated from a grammar which allows embedded phrases in the form of relative clauses, e.g. "Dog who chases cat sees girl." This experiment shows that a recurrent network can learn to represent several levels of hierarchical structure. The grammar, shown in Figure 2.17, has 23 words and one end-of-sentence marker. This grammar has a number of properties in common with natural language: number agreement, verb argument structure, interactions with relative clauses (the agent or subject in a relative clause is omitted), centre-embedding, and viable sentences (there are positions in which the end of sentence marker cannot occur).

The network architecture is shown in Figure 2.18. Local representations are used on the inputs and outputs, with two units reserved for unspecified purposes. The hidden layer consists of 70 units which are recycled to the next timestep via the context units. Local-to-distributed maps, with 10 units in the distributed representations, are used for both input and output. The training regime involves four sets of 10,000 sentences. The first set of sentences have no relative clauses and each subsequent set has a higher proportion of more complex sentences, with the fourth set having 75% complex sentences. Each set is presented to the network five times. This gradated training regime beginning with simple sentences is necessary for the network to learn the task. At the end of the training (a total of 200,000 sentences presentations) the network learned all of the above properties, i.e., it predicts verbs that agree in number with the noun, etc. Elman tested the trained network on a new set of test sentences generated in the same way as the fourth training set, and claims the network gives reasonably good predictions. However, it is difficult for the reader to evaluate how good these predictions were because although Elman does report the average error on the testing set, he does not report the average error on the training set, so there is nothing to compare it to. Also, Elman does not say how many of the sentences in

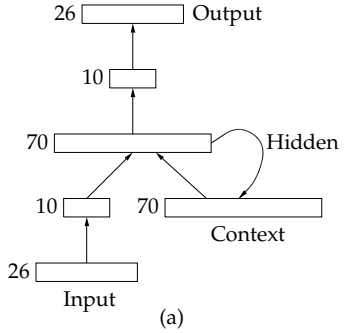


Figure 2.18: Elman's recurrent net for predicting words. (Adapted from Elman [1991])

Length	# of patterns	# of sentences	Length	# of patterns
2	6	40	10	1024
3	18	400	11	12288
4	8	128	12	36864
5	96	5120	13	10240
6	288	51200	14	122880
7	96	12288	15	368640
8	1152		16	98304
9	3456			

Table 2.1: Numbers of different sentences and sentence patterns generated by Elman's grammar. The length does not include the period.

the testing set were not actually in the training set. There are only 440 different sentences with two or three words, so it is unlikely that many of the shorter sentences in the testing set are not in the training set.

Another reason for questioning whether the network does more than rote learning is that the size of the vocabulary gives a misleading impression of the complexity of the input. There are actually only 11 different classes of vocabulary symbols (including the period) which must be distinguished in order to make optimal predictions. For example, 'boy', 'girl', 'cat', and 'dog' can all be treated as the same for the purposes of prediction. Thus, although there are 400 sentences of three words, there are only 18 different sentence patterns of this length, e.g., "singular-noun singular-transitive-verb plural-noun". When this is taken into account, the number of distinct sentence patterns is small compared to the size of the training sets, for sentences with up to 2 or 3 relative clauses. Table 2.1 lists the numbers of sentence patterns for sentences of various lengths. The network could be learning the classes of vocabulary symbols on the first training corpus, and then rote-learning sentence patterns on the later training corpora.

Elman attempted to analyze what the network was doing by looking at the trajectories of the hidden units through time. To make it possible to see these trajectories he extracted the principal components of the hidden unit space (70 dimensions) and plotted the projection of the trajectories onto the principal components. Three example trajectory projections

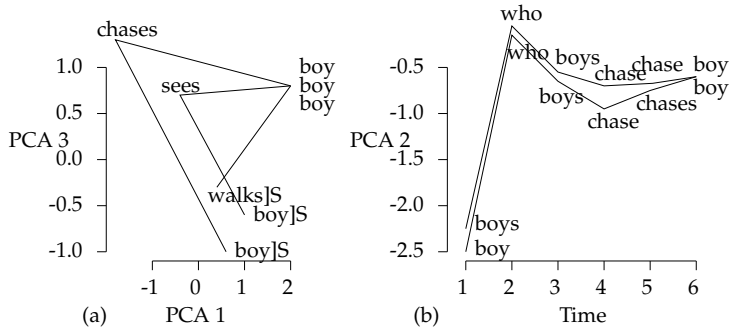


Figure 2.19: Trajectories of hidden units during processing of sentences, projected onto principal components. The final word in each sentence is marked with "]S". (Adapted from Elman [1991])

are shown in Figure 2.19a. The three verbs have different argument structure and after receiving the verb the hidden units are in a different region of state space. 'Chases' requires a direct object, 'sees' takes an optional object, and 'walks' precludes an object. One presumes that 'feeds', 'hears', and 'lives' behave similarly, but this is not reported. When a sentence can legally end, the trajectories are within a small region (at least on these two components).

Observing trajectories for sentences with centre-embedded clauses gives an opportunity for gaining some insight into how the network is representing structured state information. Figure 2.19b shows the trajectories of the hidden units while processing two similar sentences with embedded clauses: "Boys who boys chase chase boy" and "Boy who boys chase chases boy." (These sentences were not in the training corpus.) To correctly process these sentences the network must remember the number (plural or singular) of the main clause noun (the first word) while processing the intervening relative clause "who boys chase". Also, the network must ignore the number of main clause noun when processing the verb of this relative clause. To do this in general requires a stack, since an intervening clause can have more relative clauses embedded within it. Figure 2.19b does show that differences in the trajectories are maintained while the number information is relevant, but it does not explain how the network is implementing the operations of a stack. Elman states that representations seemed to degrade after about three levels of embedding, so it is possible the network did not learn to push and pop number information but merely learned the patterns for sentences with low degrees of centre-embedding.

Elman's experiments do not provide conclusive evidence that recurrent networks can learn to represent complex structure in an interesting way. His networks definitely do learn some structure, but it is unclear whether they rote-learn many sentence patterns or actually learn computational primitives for encoding and decoding hierarchical structures.

St. John and McClelland: sentence comprehension

St. John and McClelland [1990] apply recurrent networks to processing sentences. Their model concurrently learns to represent a "gestalt" meaning of the whole sentence while learning to answer questions about the sentence, based on the information in the gestalt.

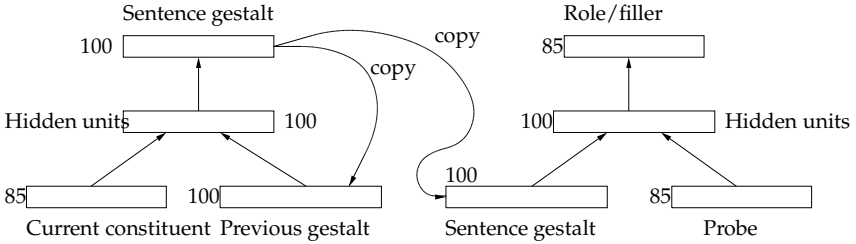


Figure 2.20: St. John and McClelland's [1990] sentence processing network.

The model is shown in Figure 2.20. It consists of two networks, one a recurrent network which builds up a gestalt of the sentence meaning as it receives the syntactic constituents of the sentence, and the other a feedforward network which decodes the gestalt into role-filler pairs. All inputs and outputs are represented in a localist fashion. Sentence constituents (simple noun phrases, prepositional phrases, and verbs) are fed to the network in sequence, and the sentence gestalt develops recursively. After the presentation of each constituent, the gestalt is probed for all role-filler pairs in the meaning of the sentence, including ones that depend upon constituents not yet presented. This forces the network to anticipate upcoming constituents and discover regularities in sentences which allow it to make good guesses about the entire meaning based on just part of the sentence. The network performs well, although training times are long, and the training corpus is small, which raises questions about how the network might scale to cope with larger corpora.

One of the motivations for this model is to see whether good representations for sentences meanings (a collection of role-filler pairs, in this case) can be learned by backpropagation. St. John and McClelland report that the conjunctive coding representation used by McClelland and Kawamoto [1986] proved unworkable, and they wanted to see if a backpropagation network could develop a more usable representation. It does seem as though the network did develop good representations for the task, but unfortunately, St. John and McClelland do not investigate the nature of this representation. Neither do they address the difficult and important problem of how recursive structure could be represented – the model only deals with sentences without embedded clauses

Sopena: parsing sentences with embedded clauses

Sopena [1991] describes a recurrent network which can process embedded sentences. He finesses the problem of representing the hierarchical structure of embedded sentences by using a sequence of simple clause frames as targets. As the network processes each word in a sentence, it outputs known information about the current clause, such as the predicate, agent, object, location, and instrument. For example, the sentence "The man that the boy was looking at was smoking" contains two simple clauses: "The man was smoking" and "The boy was looking at the man." When processing the words "the man", the first clause is the current one. Next, the second clause becomes the current one, and finally the first clause again becomes the current one.

Sopena's network is considerably more complicated than the networks used by Elman. It has two modules with context-hidden recurrences, one is for storing information about the current clause, and the other is for storing information about previous clauses. Sopena

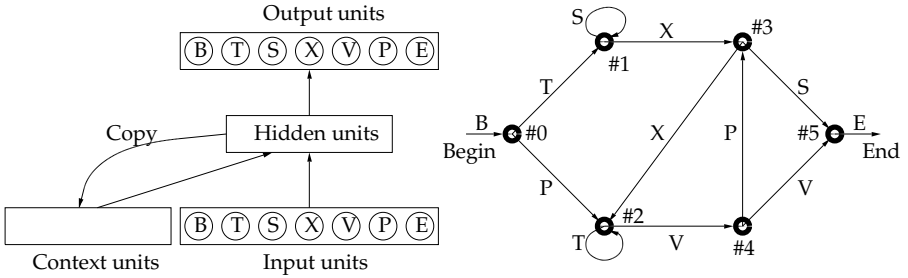


Figure 2.21: Servan-Schreiber *et al*'s grammar and recurrent network. (Adapted from Servan-Schreiber *et al* [1991].)

found that it was necessary to delete most of the recurrent connections (leaving less than 5% connectivity between context and hidden layers) in order for the network to be able learn to retain information over several timesteps. The output for network is organized as a set of case-roles. As each word is read, it must be assigned to the appropriate case-role, and output it on the appropriate set of units.

The performance of the network seems quite good. Sopena trained it on 3000 sentences for 50 epochs. He reports that the network generalized well to new sentences with both familiar and novel templates. The network only had difficulties when sentences had three or more levels of centre-embedded clauses.

Case-role assignment seems to be a better task than prediction for training a recurrent net. It requires that the network remember more information during processing, and also results in more informative error signals being fed back to the network. However, the task does not require the network to represent the entire meaning of the sentence at once; it only has to maintain a stack of incomplete clauses. The hidden layers in the network are so large (five times the width of a clause) that the network could be learning to implement a shift-register style stack. It is unclear whether the task allows for clauses to be forgotten once they are complete. The network can perform anaphora resolution, but Sopena does not give enough details to determine whether a referent lies outside of the current and incomplete clauses.

Servan-Schreiber *et al*: Graded State Machines

Servan-Schreiber, Cleeremans, and McClelland [1991] conduct a deeper investigation into the grammar learning abilities of recurrent networks like those used by Elman. They trained a network to learn a regular grammar from examples, and showed that the fully trained network has learned to be a perfect recognizer for the language. Their analysis of the acquisition of the internal representations shows that the network progressively encodes more and more temporal context during training.

Figure 2.21 shows the grammar that generates the strings, and the network used to learn the grammar. Servan-Schreiber *et al* use the same truncated error backpropagation scheme as Elman.

Servan-Schreiber *et al* investigate how well the network can carry information about distant correlations across intervening elements. They use two copies of the above finite state machine embedded in a larger finite state machine to produce strings which will test

this capability. This finite state machine produces strings of the form "$T < S > T$" and "$P < S > P$", where "$< S >$" is a string generated by the above FSM. To successfully predict the last letter of the string, the network must remember the first. Servan-Schreiber *et al* report that the network is unable to perform the task if the two embedded strings have the same characteristics. However, if they have different statistical properties (from different transition probabilities on the arcs) the network is able to learn the task. This shows the network has a tendency to forget inputs unless they are relevant to predicting the following item. This inability to learn long-range dependencies is probably due to a large extent to the truncation of the error propagation path. The network only backpropagates errors one timestep; thus there is no path for the error feedback to influence what was remembered about input presented more than one timestep before. This does not mean that the network cannot remember information if it is not needed on the next timestep; rather it means there is no pressure for it to so. As it happens, both Elman's and Servan-Schreiber *et al*'s networks often do retain information across several timesteps – it is difficult to quickly wipe out all traces of previous input in a machine with a continuous high-dimensional state. Moreover, when input is slightly relevant to the processing of intervening input, the network can learn to preserve information about it for a large number of timesteps. Servan-Schreiber *et al* suggest that natural language might have the property that the information needed to correctly process long-range dependencies is also useful for processing intervening elements, and thus could be learned by this simple type of network. However, they do not present any particularly compelling evidence to support this suggestion.

Other recurrent nets for learning regular and context-free languages

A number of researchers have continued the investigation of how well recurrent networks can learn languages from examples [Watrous and Kuhn 1992; Giles et al. 1992; Das and Mozer 1994] This work shows that recurrent networks, similar to those used by Elman and Servan-Schreiber *et al*, can reliably learn small regular grammars. Some of this work uses higher-order recurrent networks, in which connections are from pairs of units to another unit. In most cases, the training procedures use correct gradients rather than those given by Elman's truncated error propagation scheme. However, there have not been any reported successes with getting simple networks to learn context-free languages. Giles, Sun and colleagues [Giles et al. 1990; Das, Giles and Sun 1992] show that a higher-order recurrent network with an external stack can learn simple deterministic context-free grammars (e.g., parenthesis matching, $a^n b^n$, $a^n b^n c b^n a^n$). While it is interesting that a neural network can learn to use the stack, this avoids the problem of how more complex state information can be represented in a distributed representation in the hidden layer of a recurrent net.

2.4 Distributed connectionist models for explicit representation of structure

Researchers who have attempted to train networks to develop representations for complex structure have laboured under the burden of not knowing how a network can represent and manipulate structure. This creates a number of problems. It makes it difficult to analyze or explain the properties of any representations the network does develop, or to make definitive pronouncements about the how well the network will generalize to novel data or scale to larger tasks. It also forces the use of very general network architectures,

D	O	C
C	G	K
F	L	R
A	H	O
E	Q	E
P	Y	P

Table 2.2: A example receptive field table for a unit in the memory of DCPS. A unit having this table would respond to (D G R), but not (G D R).

which means that the learning procedure must conduct a long, slow search through a large function space in order to find suitable composition, manipulation and decomposition functions.

In this section I review some distributed connectionist models which deal explicitly with the problem of how compositional structure can be represented and manipulated. Only one of these models (Pollack's RAAMs) satisfies to some degree all four desiderata for Hinton's reduced descriptions: adequacy, reduction, systematicity, informativeness. The designers of these models accept the claim that connectionist models must have primitives for composing and manipulating structure if the models are to be adequate for higher-level cognitive tasks. This is in contrast to many connectionist researchers who deny this claim. For example, MacLennan [1991] comments colourfully that decomposing and recomposing symbolic structures in neural networks is "just putting a fresh coat of paint on old, rotting [symbolic] theories."

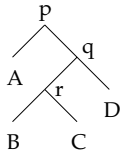
2.4.1 Touretzky and Hinton: DCPS

Touretzky and Hinton [1985; 1988] constructed a "Distributed Connectionist Production System" (DCPS) to demonstrate that connectionist networks are powerful enough to do symbolic reasoning. In later papers, Touretzky [1986a; 1986c; 1990] describes how the distributed memory of DCPS can represent variable-size tree structures.

The coarse-coded distributed memory in DCPS stores a set of triples. Each element of a triple is one of the 25 symbols, A B ... Y, so there are 15,625 possible triples. The memory has 2000 units. Each unit has a "receptive field" of 216 triples, i.e., a particular unit is turned on when one of the 216 triples it responds to is stored in the memory. The receptive fields are constructed in a factorial fashion. Each unit has a receptive field table containing six symbols for each position in the triple. Table 2.2 shows an example receptive field. A unit responds to a triple if each symbol in the triple appears in the unit's table in the correct position.

A triple is stored in the memory by turning on the units that respond to it. On average, 28 of the 2000 units will respond to a given triple.[8] A triple is deleted from memory by turning off the units which respond to it. To test whether a triple is stored in memory we check how many of the units responding to it are active. If the triple is present in memory and no other triples have been deleted from memory, then all of the units responding to

[8]The system does not work well if many more or less than 28 units respond to some triples, as is the case with random receptive fields. Touretzky and Hinton [1988] describe how the receptive field tables were carefully chosen so as keep this number close to 28.



(A p q)

(r q D)

(B r C)

Figure 2.22: A binary tree and the triples which represent it.

that triple will be active. However, if other triples have been deleted from memory, some of the units may have been turned off. Touretzky [1990] suggests that a threshold of 75% of units should be used to decide whether a triple is present in memory. DCPS can reliably store around 6 or 7 distinct triples. The probability of error on retrieval increases with the number of triples stored, and with similarity among the stored triples.

This triple memory is used as the working memory in DCPS. DCPS has four other groups of units which serve to represent rules, extract clauses, and bind variables. The rules DCPS works with are all in the following form:

Rule-2: (=x A B) (=x C D) $\rightarrow$ +(=x E F) +(P D Q) $-$(=x S T)

Rules can have one variable (here "=x"), which, when it appears in a triple, must appear in the first position. This rule says that if there are triples matching (=x A B) and (=x C D) (which would match (F A B) and (F C D), but not (F A B) and (G C D)), then two triples should be added to the working memory, and one deleted, with the appropriate variable substitutions.

Touretzky [1986a; 1986c; 1990] later designed BoltzCONS, a system which manipulates LISP-like structures (i.e., binary trees). This system uses the same triple memory to represent the structures, and has much additional machinery to interpret rules and modify memory contents. The tree representation is straightforward – each internal node in the structure is represented by a triple: left-contents, label, and right-contents. The left and right contents can be another label or an atomic symbol. Node labels function like pointers or addresses. Figure 2.22 shows a example tree and the set of triples that represent it.

The label of a triple is equivalent to the address of, or pointer to, a CONS cell in LISP memory. One advantage of this representation over a conventional pointer-based one is that the memory for triples is associative, so one can retrieve triples given any of the elements, not just the label. This allows traversal of binary trees without additional memory (a stack is usually required to traverse a binary tree).

This representation for structure has two of the four desiderata for reduced representations. The label is an adequate representation, in that one can recover the structure given the label, and is reduced. However, the label is not systematically related to the contents or structure of the tree it refers to, and gives no information about the same. It is not an explicit representation of either the contents or structure of binary trees. To find out anything about a subtree one must chase labels instead of pointers.

2.4.2 Pollack: RAAMs

Pollack [1990] uses backpropagation to learn reduced representations for trees. He sets up an auto-encoder net to learn to compress the fields of a node to a label, and uncompress the

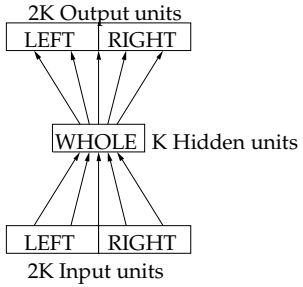


Figure 2.23: A Recursive Auto-Associative Memory (RAAM) for binary trees. The "WHOLE" is the code for an internal node in a tree, and "LEFT" and "RIGHT" can be codes for either internal or external nodes. (Adapted from Pollack [1990].)

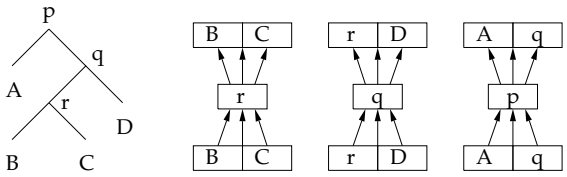


Figure 2.24: A simple tree and the auto-associations that encode it in a RAAM.

label to the fields. Pollack calls this type of network a Recursive Auto Associative Memory (RAAM).

Figure 2.23 shows the architecture of the network for binary trees, and Figure 2.24 shows the three auto-associations the network must learn to encode a simple tree. The codes for terminal nodes (A, B, C, and D) are supplied to the network, but the network must learn suitable codes for the internal nodes (p, q, and r). The training is a moving target problem, because when the weights are adjusted for one example (e.g., for (B C) $\rightarrow$ r $\rightarrow$ (B C)), this changes the representation for r, which changes the target for another example. This can cause instability in the training, so the learning rate must be kept small, which results in long training times.

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

node and is not decoded further.

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

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

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

The requirement that RAAMs be trained is both good and bad. The benefit is that RAAMs have the potential to learn to use their resources to represent, efficiently and reliably, the types of structures and terminals that are commonly encountered. The drawback is that a RAAM may be unable to represent some unfamiliar structures and terminals.

RAAMs do possess, to some extent, all four essential characteristics of reduced representations – they are adequate (in some cases), systematic (all have the same compression and expansion mapping), reduced, and informative. However, as a representation for complex structure, RAAMs have a number of problems. One problem is with the types of structures RAAMs can represent. RAAMs have a fixed number of fixed roles and there is no way of representing similarity between roles. Another problem is the lengthy training RAAMs require. Another problem is with the amount of information stored in a reduced description – as trees get deeper, more information must be stored in a fixed-size set of units, whose activations are presumably of limited precision. Some method of chunking subtrees is needed.[9] The most serious problem is that the generalization and scaling properties are largely unknown but do not appear to be terribly good. The uncertainty about them is partly due to not knowing the method by which the network compresses a full representation into a reduced one.

[9] MacLennan's invocation of Brouwer's theorem is relevant to this.

2.4.3 Smolensky: Tensor products

Smolensky [1990] describes how tensor product algebra provides a framework for the distributed representation of recursive structure. Tensor products are a way of binding multiple vectors together. They can be seen as generalized outer products or conjunctive codes (Section 2.2.3). Given two n-dimensional column-vectors $\mathbf{x}$ and $\mathbf{y}$, the second-order tensor product $T = \mathbf{x} \otimes \mathbf{y}$ is equivalent to the outer product $\mathbf{x}\mathbf{y}^{\mathrm{T}}$. T has n^2 elements, and $T_{ij} = x_i y_j$. Both lower-order and higher-order tensors exist: a first-order tensor is just a vector, and a zero'th-order tensor is a scalar. A third-order tensor is the tensor product of three vectors: $T = \mathbf{x} \otimes \mathbf{y} \otimes \mathbf{z}$, where $T_{ijk} = x_i y_j z_k$ and T has n^3 elements. Higher-order tensor products are constructed in a similar fashion – a k'th-order tensor is the product of k n-dimensional vectors and has n^k elements. The "rank" of a tensor is another name for its order. Tensor products can be built up iteratively – multiplying a third-order tensor and a second-order tensor gives a fifth-order tensor. Tensor products can be decoded by taking inner products, e.g., if $T = \mathbf{x} \otimes \mathbf{y} \otimes \mathbf{z}$, then $T \cdot (\mathbf{x} \otimes \mathbf{y}) = \mathbf{z}$ (under appropriate normalization conditions on the vectors). The inner product can be taken along any combination of dimensions. For example, given a third-order tensor $\mathbf{x} \otimes \mathbf{y} \otimes \mathbf{z}$, we can take the inner product with a second-order tensor in three different ways: along the first and second dimensions to extract $\mathbf{z}$, along the first and third dimensions to extract $\mathbf{y}$, or along the second and third dimensions to extract $\mathbf{x}$. Care must be taken to use the appropriate inner product.

Dolan and Smolensky: reimplementing DCPS with tensor products

Dolan and Smolensky [1989] use third-order tensors to reimplement Touretzky and Hinton's [1985] Distributed Connectionist Production System (DCPS). They first show how Touretzky and Hinton's triple memory can be interpreted as a rather complex, sparse, third-order tensor product representation. Next they show how a system based on straightforward tensor products is simpler and more principled. This system performs the same task as DCPS; they call it the "Tensor Product Production System" (TPPS). It has two main advantages over DCPS. One is that it easier to analyze. The other is that the representations are designed at the level of atomic symbols, combined with a general-purpose scheme for representing structured collections of these symbols. This is in contrast to the approach taken with DCPS, which was to design the representation at the level of structures. The DCPS representation is both difficult to fine-tune and limited to specific structures (binary trees).

Role-filler tensor representations for predicates

The ability to bind multiple vectors provides a number of ways of representing predicates. The simplest is the role-filler outer-product representation (second-order tensor) suggested by Smolensky [1990]. A predicate such as `p(a,b)` can be represented by the sum of role-filler products: $\mathbf{r}_1 \otimes \mathbf{a} + \mathbf{r}_2 \otimes \mathbf{b}$. Smolensky [1990] discusses two methods of decoding this role-filler binding. The first method provides exact decoding of bindings, but requires that the set of possible roles be linearly independent. It involves calculating decoding vectors $\mathbf{r}'_i$ for the roles, and then taking a tensor inner product with the binding: $(\mathbf{r}_1 \otimes \mathbf{a} + \mathbf{r}_2 \otimes \mathbf{b}) \cdot \mathbf{r}'_1 = \mathbf{a}$ (which is equivalent to the vector-by-matrix multiplication $\mathbf{r}'^{\mathrm{T}}_1(\mathbf{r}_1\mathbf{a}^{\mathrm{T}} + \mathbf{r}_2\mathbf{b}^{\mathrm{T}}) = \mathbf{a}$). The disadvantage of requiring linearly independent role vectors is that the number of roles is limited to being

less than or equal to the vector dimension, which means that the representation for roles cannot take advantage of the representational efficiency of distributed representations. The second method for decoding bindings allows the set of possible role vectors to be linearly dependent, but only provides approximate results. These results must be cleaned up by a post-processing system which finds the best match among candidate results. In this method the decoding vectors are the same as the encoding vectors: $(\mathbf{r}_1 \otimes \mathbf{a} + \mathbf{r}_2 \otimes \mathbf{b}) \cdot \mathbf{r}_1 \approx \mathbf{a}$. Care must be taken if the vectors are not normalized, as results will be scaled. Roles can also be decoded by taking an inner product in a different direction, which in matrix-vector terms corresponds to $(\mathbf{r}_1^T\mathbf{a} + \mathbf{r}_2^T\mathbf{b})\mathbf{a} \approx \mathbf{r}_1^T$.

Dolan and Smolensky [1989] suggest that predicates can be represented by a sum of third-order tensor products of predicate names, roles, and fillers. For example, their tensor product representation of `p(a,b)` would be $\mathbf{p} \otimes \mathbf{r}_1 \otimes \mathbf{a} + \mathbf{p} \otimes \mathbf{r}_2 \otimes \mathbf{b}$, where the $\mathbf{r}_1$ and $\mathbf{r}_2$ are the roles of `f`. This representation is very similar to plain role-filler bindings – it is in fact equivalent to taking the tensor product of the frame name and the set of bindings: $\mathbf{p} \otimes \mathbf{r}_1 \otimes \mathbf{a} + \mathbf{p} \otimes \mathbf{r}_2 \otimes \mathbf{b} = \mathbf{p} \otimes (\mathbf{r}_1 \otimes \mathbf{a} + \mathbf{r}_2 \otimes \mathbf{b})$. Fillers can be decoded by taking an inner product between tensors: $(\mathbf{p} \otimes \mathbf{r}_1 \otimes \mathbf{a} + \mathbf{p} \otimes \mathbf{r}_2 \otimes \mathbf{b}) \cdot (\mathbf{p} \otimes \mathbf{r}_1) \approx \mathbf{a}$. The predicate name or the role can be decoded in a similar fashion by taking an inner product in the appropriate direction.

There is not much difference between these two methods for representing predicates. Including the predicate name in the tensor product might be useful if different predicates were to share the same roles and it were desired that the representations of different predicates be distinct. However, one can always view the inclusion of the predicate name in the binding as a way of making roles of different predicates distinct – $\mathbf{p} \otimes \mathbf{r}_1$ is distinct from $\mathbf{q} \otimes \mathbf{r}_1$. With both of these methods, multiple predicates can be represented in a single tensor as a sum of bindings. However, multiple occurrences of the same predicate cause problems in both representations, because bindings from one predicate are not grouped together. For example, a single tensor representing `eat(John, fish)` and `eat(Mark, chips)` is indistinguishable from one representing `eat(John, chips)` and `eat(Mark, fish)`.

We can represent recursive structure by using higher-order tensors as fillers. The resulting tensor is of even higher-order – the order increases with each level of embedding. For example, if the tensors P and Q are second-order tensors representing predicates `p(a,b)` and `q(c,d)`, then the third-order tensor $\mathbf{r}_1 \otimes P + \mathbf{r}_2 \otimes Q$ could represent the predicate `cause(p(a,b), q(c,d))`. Although this method does provide a general scheme for representing recursive structure, it has two serious drawbacks. The first is that different components of a structure can be represented by tensors of different orders. The most immediate consequence of this is the difficulty of adding tensors of different ranks, as would be required if one filler were a predicate and another a simple object. Smolensky, Legendre and Miyata [1992] overcome this problem by multiplying lower-order tensors by "place-holder" vectors to bring all tensors up to the same rank. The second, and more severe drawback, is that the rank of the tensor, and hence the number of elements in it, grows exponentially with the depth of the structure. Smolensky *et al* [1992] suggest that this problem could be overcome by projecting higher-order tensors onto lower-order tensors. This would produce a reduced description of the higher-order tensor. However, they do not discuss what sort of projections would be suitable, or what the encoding and decoding properties would be. The danger with using projections onto lower-dimensional spaces is that the information in the null space of the projection is completely lost. If the difference between two structures is in this null space, the two structures will be given the same reduced description.

Dolan: CRAM

Dolan [1989] describes a story understanding system called 'CRAM' which uses a role-filler tensor product representation scheme. CRAM builds a representation of a simple story and finds the story schema in memory which best matches it. CRAM takes variable bindings into account during the matching process, and can perform simple reasoning about the story by rebinding variables in predicates from the retrieved schema.

CRAM uses a superimposed third-order tensor product representation (roles, fillers, and predicate names) for stories and story schemas. Each story schema in long-term memory has specific hardware to perform variable matching. It is necessary to do variable matching to retrieve the correct schema because some schemas consist of the same predicates and only differ in their variable instantiation patterns, e.g., the schemas for flattery, boasting, and recommendation.

While CRAM can deal with multiple predicates, it is not able to properly handle schemas in which predicates form a hierarchical structure. CRAM does not have a satisfactory method for forming reduced descriptions of predicates. Predicate names can be used as fillers, but this is not an adequate representation if a predicate occurs more than once in a schema. In any case, the static binding hardware used in the retrieval process is not able to properly match nested predicate structure.

Halford *et al*: component product representation for predicates

Halford, Wilson, Guo, Gayler, Wiles, and Stewart [to appear] propose a different way of using tensor products to represent predicates. They want a memory system which can store multiple predicates and retrieve one component of a predicate (the name or a filler) given its remaining components. They represent a predicate by the tensor product of all the components of the predicate. The role of a filler determines its position in the product. For example, the representation of `mother-of(woman, baby)` is **mother** $\otimes$ **woman** $\otimes$ **baby**. They represent a collection of predicates by the sum of tensors for individual predicates. This representation allows them to solve analogy problems of the form "Mother is to baby as mare is to what?" For example, suppose T is a sum of predicates, including `mother-of(woman, baby)` and `mother-of(mare, foal)`. The analogy can be solved by first taking the appropriate inner product between the tensors T and **woman** $\otimes$ **baby**,[10] yielding **mother** (provided T does not contain other predicates relating `mother` and `baby`). Having discovered that the relevant predicate is `mother-of` we next take the inner product between **mother** $\otimes$ **mare** and T, yielding the answer **foal**.

Halford *et al* propose this alternative representation for both psychological and computational reasons. The psychological reasons have to do with their claim that people can process approximately four independent dimensions in parallel. Consequently, they use tensors with a maximum order of four. The computational reason stems from their belief that it is impossible to retrieve one component of a predicate given the others predicates are represented as a sum of role-filler bindings. This is true if multiple predicates are stored as superposition of predicates. However, this has more to do with the inadequacy of superposition as a method for representing multiple predicates than with the adequacy of the role-filler representation for single predicates. Role-filler bindings are an adequate representation for this task if predicates are stored as chunks in an auto-associative memory,

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

as I show in Appendix I. Storing predicates as chunks resolves the ambiguity about which bindings come from the same predicate and actually results in a more versatile memory system – a predicate can be retrieved given any number of its components (provided the components distinguish a particular predicate). Role-filler representations have two other advantages over component product representations: they do not necessitate the imposition of an order on the components of a predicate and they have no difficult dealing with missing arguments.

Halford *et al* do not give much consideration to how recursive structure might be represented. They do mention chunking as a way of coping with more than four dimensions and with nested predicates, but do not say how vectors representing chunks, which would be akin to reduced representations, might be constructed.

2.4.4 Convolution-based models

The outer product is not the only operation on which a distributed associative memory can be based. Before matrix memories were conceived of, light holography was proposed as an analogy for human memory. A number of authors, including Gabor [1968a], Willshaw, Buneman and Longuet-Higgins [1969], and Borsellino and Poggio [1973], considered distributed associative memory models based on convolution and correlation (the mathematical operations underlying explanations of holography).

More recently, Liepa [1977], Murdock [1982; 1983; 1993], and Metcalfe [1982; 1985; 1991] have used convolution-based memories for qualitative and quantitative modelling of human memory performance. Slack [1984b; 1984a; 1986] proposed a convolution-based representation for parse trees.

Discrete convolution combines two vectors into one. Like the outer product, it can be used to bind (associate) two vectors. Suppose $\mathbf{x}$ and $\mathbf{y}$ are two n-dimensional vectors. For notational convenience I assume n that is odd and that vector indices are centred about zero, so $\mathbf{x} = (x_{-(n-1)/2}, \ldots, x_{(n-1)/2})$ and $\mathbf{y} = (y_{-(n-1)/2} \ldots y_{(n-1)/2})$. The convolution operation, denoted by "$*$", is defined as

$$\mathbf{z} = \mathbf{x} * \mathbf{y} \quad \text{where} \quad z_j = \sum_{k=-(n-1)/2}^{(n-1)/2} x_k y_{j-k} \quad \text{for } j = -(n-1) \text{ to } n-1$$

Like the outer product, convolution stores information about the association of $\mathbf{x}$ and $\mathbf{y}$ in a distributed fashion over the elements of $\mathbf{z}$. Unlike the outer product, convolution is commutative, i.e., $\mathbf{x} * \mathbf{y} = \mathbf{y} * \mathbf{x}$.[11] Convolution can in fact be viewed as a compression of the outer product. Figure 2.25 shows the outer product of two 3-dimensional vectors, and Figure 2.26 shows how the elements of the outer product are summed to form the convolution. Convolution does increase the dimensionality of vectors, but not as drastically as the outer product: $\mathbf{z} = \mathbf{x} * \mathbf{y}$ has $2n-1$ elements. Convolution can be applied recursively – the above definition is easily modified to handle $\mathbf{x}$ and $\mathbf{y}$ with different numbers of elements. The dimension of convolution products increases with the number of vectors in the product: $\mathbf{w} * \mathbf{x} * \mathbf{y}$ has $3n-2$ elements. In general a k-way convolution has $k(n-1)+1$ elements.[12] Convolution is associative, i.e., $(\mathbf{w} * \mathbf{x}) * \mathbf{y} = \mathbf{w} * (\mathbf{x} * \mathbf{y})$, this together with commutativity means that the order of vectors in a convolution product does not matter.

[11] The commutativity of convolution has consequences for psychological models. See Pike [1984] for a discussion.

[12] In Chapter 3 I describe a version of convolution which does not increase vector dimensionality.

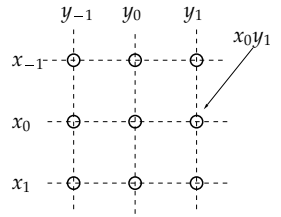


Figure 2.25: The outer product of two vectors. Each of the small circles at the intersection of a pair of lines represents a component of the outer product of $\mathbf{x}$ and $\mathbf{y}$.

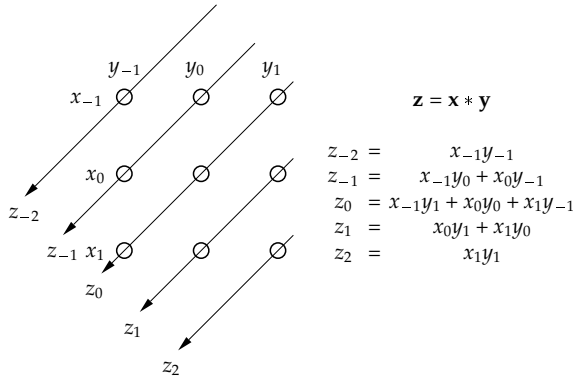


Figure 2.26: Convolution represented as a compressed outer product for $n = 3$. The convolution of $\mathbf{x}$ and $\mathbf{y}$ consists of sums (along the lines) of outer product elements. Vector indices are centred on zero for notational convenience – vectors grow in dimensionality (at both ends) with recursive convolution.

The convolution of two vectors can be decoded by correlation. The correlation operation, denoted by "#", is defined as:

$$\mathbf{w} = \mathbf{x}\#\mathbf{z} \quad \text{where} \quad w_j = \sum_{k=-(n-1)/2}^{(n-1)/2} x_{k-j} z_k \quad \text{for } j = -(n-1) \text{ to } n-1$$

Suppose $\mathbf{z} = \mathbf{x} * \mathbf{y}$. Then, under certain conditions on the elements of $\mathbf{x}$ and $\mathbf{y}$, correlating $\mathbf{x}$ with $\mathbf{z}$ reconstructs $\mathbf{y}$, with some noise:[13]

$$\mathbf{x}\#\mathbf{z} \approx \mathbf{y}$$

The noise is lower with vectors of higher dimension. A sufficient condition for this reconstruction to hold is that the elements of $\mathbf{x}$ and $\mathbf{y}$ are independently distributed as $N(0, 1/n)$ (i.e., normally distributed with mean zero and variance $1/n$). Under this condition the expected Euclidean length of vectors is 1. Under other conditions the reconstruction is

[13] The "noise" in the decoding is not stochastic noise – for a particular choice of random vectors the noise in decoding is always the same. However, over different choices of random vectors, this noise can be treated as random noise.

exact – I discuss these conditions and other properties of convolution and correlation in Chapter 3. Correlation is closely related to convolution: $\mathbf{x}\#\mathbf{z} = \mathbf{x}^* * \mathbf{z}$, where $\mathbf{x}^*$ is the mirror image of $\mathbf{x}$ (around the zero'th element), i.e., $x_i^* = x_{-i}$. Writing expressions with $\mathbf{x}^*$ and convolution instead of correlation makes them easier to manipulate, because convolution is associative and commutative, whereas correlation is neither.

As with outer products in matrix memories, convolution products can be superimposed to give a set of associations. This is often referred to as a memory *trace*, and sometimes has other information superimposed as well. The individual associations in a trace can be decoded separately, provided that the vectors in different associations are not too similar. For example, suppose $\mathbf{t} = \mathbf{x}*\mathbf{y}+\mathbf{z}*\mathbf{w}$, where the elements of $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ and $\mathbf{w}$ are independently distributed as $N(0, 1/n)$. Then, any member of a pair can be extracted from $\mathbf{t}$ given the other member of the pair, e.g.:

$$\mathbf{x}^* * \mathbf{t} \approx \mathbf{y} \quad \text{and} \quad \mathbf{w}^* * \mathbf{t} \approx \mathbf{z}.$$

The noise in the extracted vectors is quite large and increases as more associations are added to the trace. The signal-to-noise ratio is usually less than 1 even when there is only one association in the trace. However, if the dimension of the vectors is sufficiently large, extracted vectors can be reliably recognized and cleaned up. To recognize vectors we need a measure of similarity. The dot-product (vector inner product) is usually used for this. The dot-product is defined as:

$$\mathbf{x} \cdot \mathbf{y} = \sum_i x_i y_i$$

If $\mathbf{x}$ and $\mathbf{y}$ are chosen so that elements are independently distributed as $N(0, 1)$, then the expected value of $\mathbf{x} \cdot \mathbf{x}$ is one, and the expected value of $\mathbf{x} \cdot \mathbf{y}$ is zero. If the vectors are normalized (so that $\mathbf{x} \cdot \mathbf{x} = 1$) the dot-product is equal to the cosine of the angle between them. To recognize and clean up extracted vectors we must have all vectors stored in an error-correcting clean-up memory. The clean-up memory should find and output the most similar matching vector for a given input (i.e., the one with the highest dot-product). For example, we clean up the result $\mathbf{y}'$ of extracting what is bound to $\mathbf{x}$ in $\mathbf{t} = (\mathbf{x} * \mathbf{y} + \mathbf{z} * \mathbf{w})$ ($\mathbf{y}' = \mathbf{x}^* * \mathbf{t}$) by passing $\mathbf{y}'$ to the clean-up memory. If we are lucky (and if the dimension is high enough the odds will be good) the vector in clean-up memory most similar to $\mathbf{y}'$ will be $\mathbf{y}$, and $\mathbf{y}$ can be considered the result of the clean-up operation. This recognition process is the "matched filter" used by Anderson [1973] to identify components of a superposition of random vectors.

Higher-order convolutions can be decoded in a similar manner. For example, if $\mathbf{t} = \mathbf{x} * \mathbf{y} * \mathbf{z}$, then pairwise convolutions or items can be extracted from it:

$$\mathbf{x}^* * \mathbf{t} \approx \mathbf{y} * \mathbf{z}, \quad (\mathbf{x} * \mathbf{y})^* * \mathbf{t} \approx \mathbf{z}, \quad (\mathbf{x} * \mathbf{z})^* * \mathbf{t} \approx \mathbf{y}.$$

Convolution is a versatile operation for associating vectors. Associations of different orders, including plain, unconvolved vectors, can be superimposed in a single memory trace. Although associations of different orders have different numbers of elements, lower-order associations can be padded with zeros. Generally, associations of different orders do not interfere with each other – the expected similarity (dot-product) of $\mathbf{x}$ and $\mathbf{x} * \mathbf{y}$ is zero.

Human memory modelling: models and tasks

Liepa [1977], Murdock [1982; 1983; 1993], and Metcalfe [1982; 1985; 1991] have proposed various models of human memory for sequences and lists of paired-associates, all based on convolution storage methods. I mention the various storage schemes here, but omit many of the details and intricacies of psychological data that motivate differences among them. For the sake of simplicity, I just give an example of how a storage scheme stores a short sequence or small set of pairs, unless it is unclear how the scheme generalizes to longer sequences or lists. Most of these schemes have scalar parameters which control how much weight each component is given in the trace. I omit these parameters where they are unimportant to the gross characteristics of the scheme.

These models are designed to perform various recall and recognition tasks. For example, a subject might be asked to memorize the list "cow-horse, car-truck, dog-cat, and pen-pencil" and then answer such questions as "Did 'car' appear in the list?" (recognition), or "What was 'cat' associated with?" (cued recall). In a sequence recall task the subject might be asked to recall the entire sequence, or recall what item followed another, or recognize whether an item appeared in the sequence at all. Subjects' relative abilities to perform these and other tasks under different conditions, and the types of errors they produce, give insight into the properties of human memory. Some commonly varied conditions are: the number of pairs or length of sequences, the familiarity of items, and the similarity of items (both within and across pairs).

Liepa: CADAM models

Liepa [1977] describe schemes for storing lists of paired-associates and sequences, under the banner of "CADAM" (Content-Addressable Distributed Associative Memory). His scheme for storing paired associates is unadorned pairwise convolution. For example, the two pairs $(\mathbf{a}, \mathbf{b})$ and $(\mathbf{c}, \mathbf{d})$ are stored as:

$$\mathbf{T} = \mathbf{a} * \mathbf{b} + \mathbf{c} * \mathbf{d}$$

It is easy to model cued recall ("What was paired with $\mathbf{b}$?") with this scheme, but it is not so easy to do other things such as recognition ("Did $\mathbf{a}$ appear in the list"), because there is no information about individual items in the trace. Liepa's scheme for sequences is more interesting, each item is stored by associating it with the convolution of all previous items. The sequence **abcd** is stored as:

$$\mathbf{T} = \mathbf{a} + \mathbf{a} * \mathbf{b} + \mathbf{a} * \mathbf{b} * \mathbf{c} + \mathbf{a} * \mathbf{b} * \mathbf{c} * \mathbf{d}$$

This scheme supports sequential recall in the following manner:

$$\begin{aligned}
\mathbf{T} &= \underline{\mathbf{a}} + \mathbf{a} * \mathbf{b} + \mathbf{a} * \mathbf{b} * \mathbf{c} + \mathbf{a} * \mathbf{b} * \mathbf{c} * \mathbf{d} &&\approx \mathbf{a} \\
\mathbf{a}^* * \mathbf{T} &= \mathbf{a}^* * \mathbf{a} + \underline{\mathbf{a}^* * \mathbf{a} * \mathbf{b}} + \mathbf{a}^* * \mathbf{a} * \mathbf{b} * \mathbf{c} + \mathbf{a}^* * \mathbf{a} * \mathbf{b} * \mathbf{c} * \mathbf{d} &&\approx \mathbf{b} \\
(\mathbf{a} * \mathbf{b})^* * \mathbf{T} &= (\mathbf{a} * \mathbf{b})^* * \mathbf{a} + (\mathbf{a} * \mathbf{b})^* * \mathbf{a} * \mathbf{b} + \underline{(\mathbf{a} * \mathbf{b})^* * \mathbf{a} * \mathbf{b} * \mathbf{c}} + \ldots &&\approx \mathbf{c} \\
(\mathbf{a} * \mathbf{b} * \mathbf{c})^* * \mathbf{T} &= (\mathbf{a} * \mathbf{b} * \mathbf{c})^* * \mathbf{a} + \ldots + \underline{(\mathbf{a} * \mathbf{b} * \mathbf{c})^* * \mathbf{a} * \mathbf{b} * \mathbf{c} * \mathbf{d}} &&\approx \mathbf{d}
\end{aligned}$$

The decoding cues must be built up out of retrieved items as the sequence is unravelled. Each of these equations holds (approximately) because the underlined terms, such as $\mathbf{a}^* * \mathbf{a} * \mathbf{b} (\approx \mathbf{b})$, are similar to items in the clean-up memory. The other terms, such as $\mathbf{a}^* * \mathbf{a}$ and $\mathbf{a}^* * \mathbf{a} * \mathbf{b} * \mathbf{c}$, are not likely to be similar to anything in clean-up memory, assuming all the vectors have been chosen randomly. This scheme is not subject to the obvious pitfall of chaining schemes, which is that repeated items could cause jumps or loops. However, Liepa's scheme does not support sequential decoding starting from a named item or subsequence, or item recognition without decoding. This makes it somewhat unsatisfactory as a model of how the human brain stores sequences, because people can perform both of these tasks quite well.

Murdock: TODAM

Murdock [1982; 1983] proposes a "Theory of Distributed Associative Memory" (TODAM) in which both item and associative information is superimposed in the memory trace. TODAM has different schemes for storing sequences and lists of paired-associates. The scheme for paired-associates [Murdock 1982] stores (**a**, **b**) and (**c**, **d**) as:

$$\mathbf{T} = \mathbf{a} + \mathbf{b} + \mathbf{a} * \mathbf{b} + \mathbf{c} + \mathbf{d} + \mathbf{c} * \mathbf{d}$$

Here, the item information is $\mathbf{a} + \mathbf{b} + \mathbf{c} + \mathbf{d}$ and the pair information is $\mathbf{a} * \mathbf{b} + \mathbf{c} * \mathbf{d}$. Murdock uses several parameters to control the relative contributions of item and pair information and the decay of the trace as more pairs are added in. This representation supports both cued recall and item recognition. The processes for these two tasks are different. Cued recall involves correlating the trace with the cue and cleaning-up the result, whereas item recognition involves calculating the dot-product of the cue and the trace and comparing the result against a threshold.

The scheme for serial information [Murdock 1983] is similar. The memory trace for the series $\{\mathbf{f}_1, \ldots, \mathbf{f}_k\}$ is built up iteratively using chaining:

$$\begin{aligned} \mathbf{M}_1 &= \mathbf{f}_1 \\ \mathbf{M}_j &= \alpha \mathbf{M}_{j-1} + \gamma \mathbf{f}_j + \omega \mathbf{f}_j * \mathbf{f}_{j-1} \quad (\text{for } j >= 2) \end{aligned}$$

where $\mathbf{M}_j$ is the representation for the sequence up to $\mathbf{f}_j$. I show the weighting parameters because correct recall of the first item in the series depends on the relative weighting of the items. The intermediate and final traces for the sequence $\{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}$ are the following:

$$\begin{aligned} \mathbf{M}_1 &= \mathbf{a} \\ \mathbf{M}_2 &= \gamma \mathbf{b} + \omega(\mathbf{b} * \mathbf{a}) + \alpha \mathbf{a} \\ \mathbf{M}_3 &= \gamma \mathbf{c} + \omega(\mathbf{c} * \mathbf{b}) + \alpha\gamma \mathbf{b} + \alpha\omega(\mathbf{b} * \mathbf{a}) + \alpha^2 \mathbf{a} \\ \mathbf{M}_4 &= \gamma \mathbf{d} + \omega(\mathbf{d} * \mathbf{c}) + \alpha\gamma \mathbf{c} + \alpha\omega(\mathbf{c} * \mathbf{b}) + \alpha^2\gamma \mathbf{b} + \alpha^2\omega(\mathbf{b} * \mathbf{a}) + \alpha^3 \mathbf{a} \end{aligned}$$

This representation can support a number of recall and recognition tasks. Item recognition is the same as for the paired-associates scheme. Whole sequences can be recognized by forming a trace for the probe sequence in the same manner and comparing it (by the dot-product) to the memorized trace. Similar sequences will have higher dot-products than dissimilar sequences. Lewandowsky and Murdock [1989] use this model to explain human subjects' performance on a number of serial learning and recall tasks.

To do serial recall we must first find the item with the highest weight (parameters must be chosen so that $\alpha^{k-1} > \gamma$). For the above trace ($\mathbf{M}_4$) this will be **a** (assuming that superimposing the items and pairs has not upset the relative weights of items). Then we follow the chain of pairs, correlating the trace with the current item to retrieve the next. The commutativity of convolution results in a minor problem in decoding. When we correlate with an interior item the result is a superposition of the previous and next items. For example, correlating $\mathbf{M}_4$ with **b** yields the superposition of **a** and **c**. The decoding process must take this into account. It seems this scheme will not be adequate for sequences which contain repeated items. The above serial recall process could jump or loop on encountering a repeated item, and there is no obvious way in which this could be fixed.

Murdock remarks that TODAM memory traces can be regarded as memory chunks. Traces can be treated as items in their own right, and composed in a hierarchical fashion.

Murdock does not give any examples, but here is how one might look. For clarity, I omit the weighting factors. Suppose we have the two chunks (traces) for the sequences $\{\mathbf{a}, \mathbf{b}\}$ and $\{\mathbf{c}, \mathbf{d}, \mathbf{e}\}$:

$$\begin{aligned} \mathbf{C}_1 &= \mathbf{a} + \mathbf{b} + \mathbf{a} * \mathbf{b} \\ \mathbf{C}_2 &= \mathbf{c} + \mathbf{d} + \mathbf{e} + \mathbf{c} * \mathbf{d} + \mathbf{d} * \mathbf{e} \end{aligned}$$

A sequence composed of these two chunks, and its expansion, would be stored as:

$$\begin{aligned} \mathbf{C} &= \mathbf{C}_1 + \mathbf{C}_2 + \mathbf{C}_1 * \mathbf{C}_2 \\ &= \mathbf{a} + \mathbf{b} + \mathbf{a} * \mathbf{b} + \mathbf{c} + \mathbf{d} + \mathbf{e} + \mathbf{c} * \mathbf{d} + \mathbf{d} * \mathbf{e} \\ &\quad + (\mathbf{a} + \mathbf{b} + \mathbf{a} * \mathbf{b}) * (\mathbf{c} + \mathbf{d} + \mathbf{e} + \mathbf{c} * \mathbf{d} + \mathbf{d} * \mathbf{e}) \end{aligned}$$

This brings up the possibility that TODAM traces could function as a reduced representation. The one problem with this is that the dimension of vectors grows with recursive convolution – not as fast as tensor products grow, but they still grow. It is not inconceivable that some chunks could be nested quite a few levels deep, and it is somewhat inelegant (and impractical) for vectors representing high-level chunks to be several times wider than vectors representing low-level chunks.

Murdock [1983] briefly discusses how predicates might be represented in the TODAM model. He suggests three different ways of storing the predicate `p(a,b)`:

$$\begin{aligned} \mathbf{P}_1 &= \mathbf{p} * \mathbf{a} + \mathbf{a} * \mathbf{b} \\ \mathbf{P}_2 &= \mathbf{p} * \mathbf{a} + \mathbf{p} * \mathbf{b} \\ \mathbf{P}_3 &= \mathbf{p} * \mathbf{a} * \mathbf{b} \end{aligned}$$

The first two of these methods are simple chaining representations, the first in the order $\{\mathbf{p}, \mathbf{a}, \mathbf{b}\}$ and the second in the order $\{\mathbf{a}, \mathbf{p}, \mathbf{b}\}$. The third stores the predicate as a triple. None of these methods is entirely satisfactory. The first and second method assume some ordering on the arguments of a predicate. The second and third methods lose argument order information – `p(a,b)` and `p(b,a)` have the same representation. With the third method it is difficult to retrieve the arguments knowing only the predicate name – it is only easy to decode one argument if we already know the predicate name and one other argument. All three methods fare poorly if some arguments are optional – the retrieval processes for a particular argument changes according to which other arguments are present. These problems are not insurmountable, but as Murdock points out, there are many ways predicates can be represented using convolution, and other ways could be superior.

Metcalfe: CHARM

Metcalfe [1982; 1985; 1991] proposes another convolution-based representation for paired-associates lists. She uses this representation in her "Composite Holographic Associative Recall Model" (CHARM). CHARM differs from Murdock's TODAM in that item information is stored in the memory trace in the form of auto-convolutions. For example, the pairs (**a**, **b**) and (**c**, **d**) are stored as:

$$\mathbf{T} = \mathbf{a} * \mathbf{a} + \mathbf{b} * \mathbf{b} + \mathbf{a} * \mathbf{b} + \mathbf{c} * \mathbf{c} + \mathbf{d} * \mathbf{d} + \mathbf{c} * \mathbf{d}$$

The significance of this is that the recognition process is nearly identical to the recall process. Testing whether **a** is present in the trace involves correlating the trace with **a**, and computing similarity of the result to **a**. This leads to CHARM and TODAM making different predictions about the dependence of recall and recognition in the presence of similarity among items in the trace. Metcalfe [1991] claims that the data on human performance is more in line with the predictions derived from CHARM.

Murdock: TODAM2

Murdock [1992; 1993], in response to some failings of the TODAM model, proposes a more general and more complex scheme for storing sequences and lists of paired associates. Murdock calls this new model TODAM2. I will not describe TODAM2 in full – I will only explain the "chunking" idea which is a central component of it.

A "chunk" is a representation for a sequence of items. The chunk for the sequence $\{\mathbf{f}_1, .., \mathbf{f}_k\}$ is:

$$\mathbf{C}_k = \sum_{i=1}^{k} \left(\sum_{j=1}^{i} \mathbf{f}_j \right)^i,$$

where the i'th power of a vector is the vector convolved with itself i times ($\mathbf{a}^2 = \mathbf{a} * \mathbf{a}$). Some examples for the chunks of $\{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}\}$ make this clearer:

$$\begin{aligned}
\mathbf{C}_1 &= \mathbf{a} \\
\mathbf{C}_2 &= \mathbf{a} + (\mathbf{a} + \mathbf{b})^2 \\
&= \mathbf{a} + (\mathbf{a} + \mathbf{b}) * (\mathbf{a} + \mathbf{b}) \\
&= \mathbf{a} + \mathbf{a} * \mathbf{a} + 2\mathbf{a} * \mathbf{b} + \mathbf{b} * \mathbf{b} \\
\mathbf{C}_3 &= \mathbf{a} + (\mathbf{a} + \mathbf{b})^2 + (\mathbf{a} + \mathbf{b} + \mathbf{c})^3 \\
\mathbf{C}_4 &= \mathbf{a} + (\mathbf{a} + \mathbf{b})^2 + (\mathbf{a} + \mathbf{b} + \mathbf{c})^3 + (\mathbf{a} + \mathbf{b} + \mathbf{c} + \mathbf{d})^4
\end{aligned}$$

For representing sequences, TODAM2 uses one chunk for the entire sequence, unless the sequence is broken down into subsequences. For paired associates, TODAM2 uses a chunk for each pair. Chunks contain all the convolution products used in CADAM, TODAM, and CHARM, thus the access processes of these models can be used with chunks.

Murdock uses chunks for two reasons: to support recall of missing items, and to create higher-order associations, the need for which is argued by Humphreys, Bain and Pike [1989].

Recall of missing items involves recalling the items from an original list which are not in a new list. For example, given an original list $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ and a new list $\{\mathbf{a}, \mathbf{c}\}$, the missing item is **b**. This information can be extracted from the chunk for $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ by correlating it with $\mathbf{a} * \mathbf{c}$:

$$(\mathbf{a} * \mathbf{c})^* * \mathbf{C} = (\mathbf{a} * \mathbf{c})^* * (\mathbf{a} + (\mathbf{a} + \mathbf{b})^2 + (\mathbf{a} + \mathbf{b} + \mathbf{c})^3)$$

Most of the terms in the expansion of this turn out to be noise, except for:

$$(\mathbf{a} * \mathbf{c})^* * (6\mathbf{a} * \mathbf{b} * \mathbf{c} + 3\mathbf{a} * \mathbf{c} * \mathbf{c} + 3\mathbf{a} * \mathbf{a} * \mathbf{c}) \approx 6\mathbf{b} + 3\mathbf{c} + 3\mathbf{a},$$

which can be cleaned up to give **b**.

The problem with Murdock's chunks is that they introduce many higher-order associations for which there is no obvious need, such as $\mathbf{a}^2 * \mathbf{b}$. These dilute the useful associations,

which has the effect of adding noise to decoded vectors. This means that very a high vector dimension is required to achieve acceptable levels of reliability. The problem is particularly bad when chunks are composed of more than three items. One wonders whether the reasons for using chunks could be satisfied by some other more economical storage scheme. For example, another way to do missing item recall is to compute the difference of superpositions, as do Hadley, Healy and Murdock [1992]:

$$(\mathbf{a} + \mathbf{b} + \mathbf{c}) - (\mathbf{a} + \mathbf{c}) = \mathbf{b}$$

Doing missing item recall in this way does not require any more convolution products or vectors than are already required for other tasks such as item recognition.

Chapter 3

Holographic Reduced Representations

All of the schemes for the distributed representation of hierarchical structure reviewed in the previous chapter have some flaws. In the remainder of this thesis I present and discuss a scheme, called "Holographic Reduced Representations" (HRRs) which I believe overcomes these flaws. HRRs support the useful properties of distributed representations that were identified in Section 1.4.2: explicitness, explicit similarity, redundancy, continuity, and efficiency. They satisfy Hinton's four desiderata for reduced representations: they are representationally adequate, the reduced descriptions are represented over fewer units, reduced and full descriptions are related in a systematic manner, and the reduced descriptions give some information about the contents of the full description.

HRRs incorporate ideas and techniques from several fields of research. HRRs are an concrete implementation of Hinton's [1990] notion of "reduced descriptions". HRRs use the role-filler representation for predicates suggested by Hinton, McClelland and Rumelhart [1986], but bind roles and fillers with convolution instead of the outer product. HRRs are based on convolution, like Murdock [1982] and Metcalfe's [1982] models, but use a version of convolution (circular convolution) which does not increase the dimensionality of vectors. Hierarchical structure is represented by recursively composing role-filler bindings in similar manner to Smolensky's [1990] recursively composed tensor products, but unlike the tensor product, circular convolution does not increase vector dimensionality.

HRRs are constructed in a simple and regular manner, as with Smolensky's tensor products (to which HRRs are closely related). The simple method of construction makes it possible to analyze and predict the properties of HRRs, and allows us to make definitive statements about how well HRRs will scale to larger systems.

The biggest advantage of HRRs is that they make it easy to represent structure. This means that a learning system which uses HRRs does not need to expend computational resources learning how to represent structure – it is free to concentrate on the truly difficult task of learning the important relationships among objects in the task.

3.1 Circular convolution and correlation

Tensor products and ordinary convolution are recursively applicable associative operators, which makes them candidates for the representation of hierarchical structure. However, both increase the dimensionality of vectors, which makes them somewhat impractical

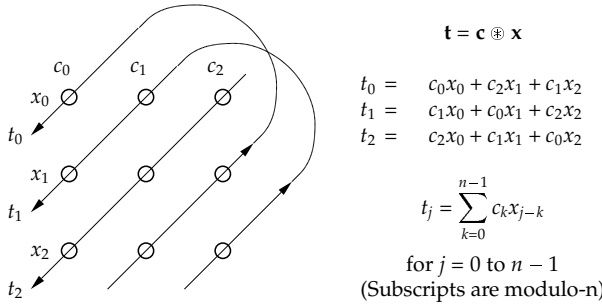


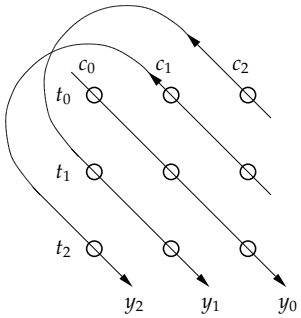
Figure 3.1: Circular convolution ($\circledast$) represented as a compressed outer product for $n = 3$. The circles represent the elements of the outer product of $\mathbf{x}$ and $\mathbf{c}$. The elements of the convolution are the sum, along the wrapped diagonals, of the elements of the outer product.

for this purpose (although convolution increases dimensionality far less than the tensor product). *Circular convolution* is a version of convolution which does not increase vector dimensionality. It is well known in signal processing (sometimes as "wrapped" convolution, e.g., Gabel and Roberts [1973]) but its only use in associative memories has been by Willshaw, Buneman and Longuet-Higgins [1969] in their "non-linear correlograph". The result of the circular convolution of two vectors of n elements has just n elements – there is no expansion of dimensionality. Like ordinary convolution, circular convolution can be regarded as a compression of the outer product of two vectors, as illustrated in Figure 3.1.

Circular convolution can be regarded as a multiplication operator for vectors. It has many properties in common with both scalar and matrix multiplication. It is commutative: $\mathbf{x} \circledast \mathbf{y} = \mathbf{y} \circledast \mathbf{x}$; associative: $\mathbf{x} \circledast (\mathbf{y} \circledast \mathbf{z}) = (\mathbf{x} \circledast \mathbf{y}) \circledast \mathbf{z}$; and bilinear: $\mathbf{x} \circledast (\alpha\mathbf{y} + \beta\mathbf{z}) = \alpha\mathbf{x} \circledast \mathbf{y} + \beta\mathbf{x} \circledast \mathbf{z}$. There is an identity vector: $\bar{I} \circledast \mathbf{x} = \mathbf{x}$, $\bar{I} = [1, 0, 0, \ldots]$; and a zero vector: $\bar{0} \circledast \mathbf{x} = \bar{0}$ and $\mathbf{x} + \bar{0} = \mathbf{x}$, $\bar{0} = [0, 0, 0, \ldots]$. For nearly all vectors $\mathbf{x}$, an exact inverse exists: $\mathbf{x}^{-1} \circledast \mathbf{x} = \bar{I}$. For vectors from certain distributions, there is a stable approximate inverse which is generally more useful than the exact inverse. In algebraic expressions I give convolution the same precedence as multiplication, so that $\mathbf{x} \circledast \mathbf{y} + \mathbf{z} = (\mathbf{x} \circledast \mathbf{y}) + \mathbf{z}$, and higher precedence than the dot-product, so that $\mathbf{x} \circledast \mathbf{y} \cdot \mathbf{w} \circledast \mathbf{z} = (\mathbf{x} \circledast \mathbf{y}) \cdot (\mathbf{w} \circledast \mathbf{z})$.

There is also a circular version of correlation, which under certain conditions is an approximate inverse of circular convolution. Circular correlation, $\circledcirc$, can also be regarded as a compression of the outer product, as illustrated in Figure 3.2. Suppose we have a trace which is the convolution of a cue with another vector, $\mathbf{t} = \mathbf{c} \circledast \mathbf{x}$. Correlation of $\mathbf{c}$ with $\mathbf{t}$ reconstructs a distorted version of $\mathbf{x}$: $\mathbf{y} = \mathbf{c} \circledcirc \mathbf{t}$ and $\mathbf{y} \approx \mathbf{x}$.

Multiple associations can be represented by the sum of the individual associations. Upon decoding, the contribution of the irrelevant terms can be ignored as distortion. For example, if $\mathbf{t} = \mathbf{c}_1 \circledast \mathbf{x}_1 + \mathbf{c}_2 \circledast \mathbf{x}_2$, then the result of decoding of $\mathbf{t}$ with $\mathbf{c}_1$ is $\mathbf{c}_1 \circledcirc \mathbf{c}_1 \circledast \mathbf{x}_1 + \mathbf{c}_1 \circledcirc \mathbf{c}_2 \circledast \mathbf{x}_2$. If the vectors have been chosen randomly the second term will, with high probability, have low correlation with all of $\mathbf{c}_1, \mathbf{c}_2, \mathbf{x}_1$ and $\mathbf{x}_2$ and the sum will be recognizable as a distorted version of $\mathbf{x}_1$.



$$\mathbf{y} = \mathbf{c} \circledast \mathbf{t}$$

$$\begin{aligned} y_0 &= c_0t_0 + c_1t_1 + c_2t_2 \\ y_1 &= c_2t_0 + c_0t_1 + c_1t_2 \\ y_2 &= c_1t_0 + c_2t_1 + c_0t_2 \end{aligned}$$

$$y_j = \sum_{k=0}^{n-1} c_k t_{k+j}$$

for $j = 0$ to $n-1$
(Subscripts are modulo-n)

Figure 3.2: Circular correlation ($\circledast$) represented as a compressed outer product for $n = 3$.

3.1.1 Distributional constraints on the elements of vectors

A sufficient condition for correlation to decode convolution is that the elements of each vector (of dimension n) be independently and identically distributed with mean 0 and variance $1/n$. This results in the expected Euclidean length of a vector being 1. Examples of suitable distributions for elements are the normal distribution and the discrete distribution with values equiprobably $\pm 1/\sqrt{n}$. The reasons for these distributional constraints should become apparent in the next subsection.

The tension between these constraints and the conventional use of particular elements of vectors to represent meaningful features in distributed representations is discussed in Section 3.5.

3.1.2 Why correlation decodes convolution

It is not immediately obvious why correlation decodes convolution. However, it is not hard to see if an example is worked through. Consider vectors with three elements, $\mathbf{c} = (c_0, c_1, c_2)$ (the cue), and $\mathbf{x} = (x_0, x_1, x_2)$, where the c_i and x_i are independently drawn from $\mathrm{N}(0, 1/n)$ (i.e., a normal distribution with mean 0 and variance $1/n$, $n = 3$ in this example). The convolution of $\mathbf{c}$ and $\mathbf{x}$ is:

$$\mathbf{c} \circledast \mathbf{x} = \begin{bmatrix} c_0x_0 + c_1x_2 + c_2x_1 \\ c_0x_1 + c_1x_0 + c_2x_2 \\ c_0x_2 + c_1x_1 + c_2x_0 \end{bmatrix}$$

We can reconstruct $\mathbf{x}$ from this trace by correlating with the cue $\mathbf{c}$:

$$\mathbf{c} \circledast (\mathbf{c} \circledast \mathbf{x})$$

$$= \begin{bmatrix} x_0(c_0^2 + c_1^2 + c_2^2) + x_1c_0c_2 + x_2c_0c_1 + x_1c_0c_1 \\ \qquad + x_2c_1c_2 + x_1c_1c_2 + x_2c_0c_2 \\ x_1(c_0^2 + c_1^2 + c_2^2) + x_0c_0c_1 + x_2c_0c_2 + x_0c_0c_2 \\ \qquad + x_2c_1c_2 + x_0c_1c_2 + x_2c_0c_1 \\ x_2(c_0^2 + c_1^2 + c_2^2) + x_0c_0c_1 + x_1c_1c_2 + x_0c_1c_2 \\ \qquad + x_1c_0c_2 + x_0c_0c_2 + x_1c_0c_1 \end{bmatrix}$$

$$= \begin{bmatrix} x_0(1+\zeta) + \eta_0 \\ x_1(1+\zeta) + \eta_1 \\ x_2(1+\zeta) + \eta_2 \end{bmatrix} = (1+\zeta)\mathbf{x} + \bar{\eta}$$

where ζ and the η_i can be treated as zero-mean noise. The variances of ζ and the η_i are inversely proportional to n. The distributions of the ζ and η_i are normal in the limit as n goes to infinity, and the approximation to the normal is excellent for n as small as 16. In general, it is safe to use the normal approximations because typical values for n in convolution-based associative memory systems are in the hundreds and thousands.

Using the central limit theorem, and assuming the c_i and x_i are independent and distributed as $\mathrm{N}(0, 1/n)$, the distributions of ζ and the η_i for large n are as follows:

$\zeta \stackrel{d}{=} \mathrm{N}(0, \frac{2}{n})$, since $\zeta = (c_0^2 + c_1^2 + \ldots + c_n^2) - 1$, and the c_i^2 are independent and have mean $1/n$ and variance $2/n^2$.

$\eta_i \stackrel{d}{=} \mathrm{N}(0, \frac{n-1}{n^2})$, since the $n(n-1)$ terms like $x_jc_kc_l$ $(k \neq l)$ have mean 0 and variance $1/n^3$, and the pairwise covariances of these terms are zero.

It is more useful to calculate the variance of the dot-product $\mathbf{x} \cdot (\mathbf{c} \circledast (\mathbf{c} \circledast \mathbf{x}))$ than the variances of the elements of $\mathbf{c} \circledast (\mathbf{c} \circledast \mathbf{x})$, because this can be used to calculate the probability of correct decoding. This in turn allows us to calculate the minimum dimension (n) for which the probability of correct decoding associations is acceptable. However, calculating these variances is not simple, because one must take into account the covariances of the noise terms in the different elements. Extensive tables of variances for dot-products of various convolution products have been compiled by Weber [1988] for unwrapped convolution. Unfortunately, these do not apply exactly to circular convolution. The means and variances for dot-products of some common circular convolution products will be given in Table 3.1 in Section 3.6.1.

3.1.3 Relationship of convolution to correlation

The correlation of $\mathbf{c}$ and $\mathbf{t}$ is equivalent to the convolution of $\mathbf{t}$ with the *involution* of $\mathbf{c}$ [Schönemann 1987]. The involution of $\mathbf{c}$ is the vector $\mathbf{d} = \mathbf{c}^*$ such that $d_i = c_{-i}$, where subscripts are modulo-n.[1] For example, if $\mathbf{c} = (c_0, c_1, c_2, c_3)$, then $\mathbf{c}^* = (c_0, c_3, c_2, c_1)$. Writing $\mathbf{c}^* \circledast \mathbf{t}$ is preferable to writing $\mathbf{c} \circledast \mathbf{t}$ because it simplifies algebra, since correlation is neither associative nor commutative whereas convolution is both. Involution distributes over addition and convolution, and is its own inverse:

$$(\mathbf{a} + \mathbf{b})^* = \mathbf{a}^* + \mathbf{b}^*, \quad (\mathbf{a} \circledast \mathbf{b})^* = \mathbf{a}^* \circledast \mathbf{b}^*, \quad \mathbf{a}^{**} = \mathbf{a}$$

In analogy to inverse matrices, it is sometimes convenient to refer to $\mathbf{c}^*$ as the *approximate inverse* of $\mathbf{c}$. The exact inverse of vectors under convolution (i.e., $\mathbf{c}^{-1}$) will be discussed in Section 3.6.3.

3.1.4 How much information is stored in a convolution trace?

Since a convolution trace only has n numbers in it, it may seem strange that several pairs of vectors can be stored in it, since each of those vectors also has n numbers. The reason is that the vectors are stored with very poor fidelity. The convolution trace stores enough

[1] *Involution* has a more general meaning, but in this paper I use it to mean a particular operation.

information to recognize the vectors in it, but not enough to reconstruct them accurately. For recognition, we only need to store enough information to discriminate an item from other possible items. Suppose we have M equiprobable items, each represented by an n-dimensional vector. About $2k \log_2 M$ bits of information are needed to represent k pairs of those items for the purposes of recognition.[2] The dimensionality of the vectors does not enter into this calculation, only the number of vectors matters.

For example, if we have 1024 items (each represented by a different vector), then the number of bits required to identify four pairs of those items is slightly less than $2 \times 4 \times \log_2 1024 = 80$ bits (this is a slight overestimation because the pairs are unordered). A convolution memory using random vectors with 512 elements can reliably store four pairs of these items (see Appendix D). Storing 80 bits of information in 512 floating-point numbers is not particularly efficient, but for the storage of complex structure this is not a critical issue. Also, the floating-point numbers need not be stored to high-precision. In fact, in Willshaw, Buneman and Longuet-Higgins [1969] non-linear correlograph, which is the most information-efficient convolution-based associative memory, all vector elements are 0 or 1.

3.2 Superposition Memories

One of the simplest ways to store a set of vectors is to superimpose them (i.e., add them together). In general, such storage does not allow for recall or reconstruction of the stored items, but it does allow for recognition, i.e., determining whether a particular item has been stored or not. Anderson [1973] described and analyzed a memory model based on simple superposition of random vectors.

The principle of superposition memory can be stated thus: "adding together two high-dimensional vectors gives a vector which is similar to each and not very similar to anything else."[3] This principle underlies the ability to superimpose traces in both convolution and matrix memories.

It is not necessary for elements of vectors to have continuous values for superposition memories to work. Also, their capacity can be improved by applying a suitable non-linear (e.g., threshold) function to the trace. Touretzky and Hinton [1988] and Rosenfeld and Touretzky [1988] discuss binary distributed memories, in which representations are superimposed by elementwise binary-OR. However, I do not use binary representations in this thesis because of difficulties with maintaining constant density (see Section 3.5.2).

3.2.1 The probability of correct recognition with superposition memories

It is simple to calculate the probability of correct recognition of a random vector stored in a superposition memory. I do this here because the same techniques are used to calculate the probabilities of correct reconstruction from convolution-based memories. This analysis is essentially the same as that given in Anderson [1973]. In this section I give an expression for the probability of correct recognition, but do not give an analytic solution for it, because it involves an optimization. In Appendix B, I report results for the numerical optimization of this equation. In Appendix C I give a lower bound for the solution.

Suppose we have a superposition memory with the following parameters:

[2] Actually, slightly less than $2k \log_2 M$ bits are required if the pairs are unordered.

[3] This applies to the degree that the elements of the vectors are randomly and independently distributed.

- n: the dimensions of the vectors.
- A set $\mathbb{E}$ of m vectors, $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$ etc, with elements independently distributed as $N(0, 1/n)$.
- A memory trace $\mathbf{t}$, which is the superposition of k distinct vectors from $\mathbb{E}$. For the purposes of this analysis, normalization is unimportant, so I do not normalize the trace.
- Pr(All Correct), the probability of correctly determining which vectors are and are not stored in the memory trace.
- s_a and s_r, the accept and reject signals (see below).

To test whether a particular vector $\mathbf{x}$ from $\mathbb{E}$ is in the trace $\mathbf{t}$, we compute the dot-product of $\mathbf{x}$ and $\mathbf{t}$. The resulting signal will be from one of two distributions; the accept distribution S_a (if $\mathbf{x}$ is in the trace), or the reject distribution S_r (if $\mathbf{x}$ is not in the trace). The means and variances of these distributions can be calculated by expanding $\mathbf{x} \cdot \mathbf{t}$. For example, consider the trace $\mathbf{t} = \mathbf{a} + \mathbf{b} + \mathbf{c}$, and a signal from the accept distribution:

$$s_a = \mathbf{a} \cdot \mathbf{t} = \mathbf{a} \cdot \mathbf{a} + \mathbf{a} \cdot \mathbf{b} + \mathbf{a} \cdot \mathbf{c}.$$

Consider typical vectors elements x_i and y_j, which are independently distributed as $N(0, 1/n)$. It is easy to show that $E[x_i^2] = 1/n$ and $\text{var}[x_i^2] = 2/n^2$, and $E[x_i y_j] = 0$ and $\text{var}[x_i y_j] = 1/n^2$. By the central limit theorem, the terms like $\mathbf{a} \cdot \mathbf{a}$ are distributed as $N(1, 2/n)$, and the terms like $\mathbf{a} \cdot \mathbf{b}$ are distributed as $N(0, 1/n)$. Since these terms all have zero covariance, we can add means and variances to get $s_a \stackrel{d}{=} N(1, (k+1)/n)$ and $s_r \stackrel{d}{=} N(0, k/n)$.

The value of the dot-product $\mathbf{x} \cdot \mathbf{t}$ (the signal) tells us whether or not the item $\mathbf{x}$ is present in the trace. If the signal is greater than some threshold t we assume that it is from the accept distribution and thus the item is in the trace, and if it is less we assume it is not.

Using cumulative distribution functions, we can work out the probability of correctly deciding an item was stored in a trace ($\text{Pr(Hit)} = \text{Pr}(s_a > t)$), and the probability of correctly deciding an item was not stored in a trace ($\text{Pr(Reject)} = \text{Pr}(s_r < t)$). The threshold t can be chosen to maximize the probability of correctly identifying all the items stored (and not stored) in a particular trace:

$$\begin{aligned} \text{Pr(All Correct)} &= \text{Pr(Hit)}^k \, \text{Pr(Reject)}^{m-k} && (3.1) \\ &= \max_t \; \text{Pr}(s_a > t)^k \, \text{Pr}(s_r < t)^{m-k} && (3.2) \end{aligned}$$

The probability density functions (pdfs) for s_a and s_r and the optimal single threshold are shown in Figure 3.3, for an example with $n = 64$, $m = 100$, and $k = 3$ (for which $\text{Pr(All Correct)} = 0.68$). This threshold was found by numerical optimization of the expression for Pr(All Correct). We can actually achieve better discrimination by using a double-threshold scheme. In general, the optimal scheme for deciding which of two normal distributions a signal comes from involves testing whether the signal is in a region around the distribution with the smaller variance. However, the improvement gained by using two thresholds is small when misclassification probabilities are low.

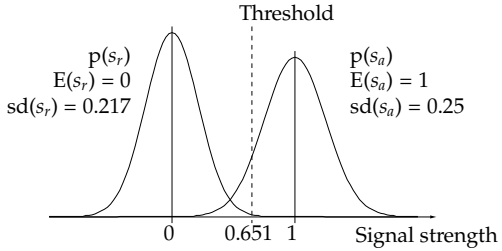


Figure 3.3: Probability density functions for accept (s_a) and reject (s_r) signals for recognition in a linear superposition memory, with $n = 64$ and $k = 3$. The threshold shown maximizes Pr(All Correct) for $m = 100$.

This decision procedure is not infallible, but n can be chosen to make the probability of error acceptably low, since the variances of the signals are inversely proportional to n. When the variances of the signals are smaller, the probability of error is lower.

This analysis here treats signal values as random variables, but their randomness is only a consequence of the random choice of the original vectors. For any particular trace with a particular set of vectors, the signal values are deterministic. This style of analysis is consistently used throughout this paper: there are no stochastic operations, only randomly chosen vectors.

3.2.2 The use of thresholds

Fixed thresholds are helpful in the analysis of probability of correct retrieval but they are not very good for determining the result of a similarity match in practice. The main reason for this is that the optimal threshold varies with both the composition of the trace and with the particular objects that comprise the trace (see Section 3.10.4 for an example). Consequently, no single fixed threshold will be appropriate for choosing the winning matches in all situations, and it is impractical to compute a new threshold for every situation. A simpler scheme is just to choose the most similar match, though this is of limited versatility. A possible enhancement of is a no-decision region; if the highest score is not more than some fixed amount greater than the next highest score, then the result is considered unclear.

3.3 The need for clean-up memories

Linear convolution memories are not able to provide accurate reconstructions. If a system using convolution representations is to do some sort of recall (as opposed to recognition), it must have an additional error-correcting auto-associative item memory. This is needed to clean up the noisy vectors retrieved from the convolution traces. This clean-up memory must store all the items that the system can produce. When given as input a noisy version of one of those items it must either output the closest item or indicate that the input is not close enough to any of the stored items. Note that one convolution trace stores only a few associations or items, and the clean-up memory stores many items.

For example, suppose the system is to store pairs of random vectors $\mathbf{a}, \mathbf{b}, \ldots, \mathbf{z}$. The clean-up memory must store these 26 vectors and must be able to output the closest item

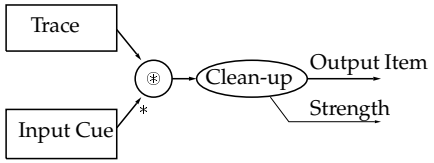


Figure 3.4: A hetero-associator machine. The "*" on the operand to the convolution indicates the approximate inverse is taken.

for any input vector (the "clean-up" operation). Such a system is shown in Figure 3.4. The trace is a sum of convolved pairs, e.g., $\mathbf{t} = \mathbf{a} \circledast \mathbf{b} + \mathbf{c} \circledast \mathbf{d} + \mathbf{e} \circledast \mathbf{f}$. The system is given one item as an input cue and its task is to output the associated item from the trace. It should also output a scalar value (the strength) which is high when the input cue was a member of a pair, and low when the input cue was not a member of a pair. When given the above trace $\mathbf{t}$ and $\mathbf{a}$ as a cue it should produce $\mathbf{b}$ and a high strength. When given $\mathbf{g}$ as a cue it should give a low strength. The item it outputs is unimportant when the strength is low.

Many of systems mentioned in Chapter 2 use some type of clean-up memory. For example, BoltzCONS uses "pull-out" networks, CRAM and DCPS use "clean-up circuits", and TODAM uses an "R-system". The exact method of implementation of the clean-up memory is unimportant. Hopfield networks are probably not a good candidate because of their low capacity in terms of the dimension of the vectors being stored. Kanerva networks [Kanerva 1988] have sufficient capacity, but can only store binary vectors. For the simulations reported in this thesis I store vectors in a list and compute all dot-products in order to find the closest match.

3.4 Representing complex structure

Pairs of items are easy to represent in many types of associative memory, but convolution memory is also suited to the representation of more complex structure, namely sequences, predicates, and recursive predicates.

3.4.1 Sequences

Sequences can be represented in a number of ways using convolution encoding. An entire sequence can be represented in one memory trace, with the probability of error increasing with the length of the stored sequence. Alternatively, chunking can be used to represent a sequence of any length in a number of memory traces.

Murdock [1983; 1987], and Lewandowsky and Murdock [1989] propose a chaining method of representing sequences in a single memory trace, and model a large number of psychological phenomena with it. The technique used stores both item and pair information in the memory trace, for example, if the sequence of vectors to be stored is **abc**, then the trace is

$$\alpha_1\mathbf{a} + \beta_1\mathbf{a} \circledast \mathbf{b} + \alpha_2\mathbf{b} + \beta_2\mathbf{b} \circledast \mathbf{c} + \alpha_3\mathbf{c},$$

where the α_i and β_i are weighting parameters, with $\alpha_i > \alpha_{i+1}$. The retrieval of the sequence begins with retrieving the strongest component of the trace, which will be $\mathbf{a}$. From there the retrieval is by chaining — correlating the trace with the current item to retrieve the

next item. The end of the sequence is detected when the correlation of the trace with the current item is not similar to any item in the clean-up memory. This representation of sequences has the properties that a sequence is similar to all of the items in it, retrieval can start from any given element of the sequence, and similar sequences will have similar representations. It has the disadvantage that some sequences with repeated items cannot be properly represented.

Another way to represent sequences is to use the entire previous sequence as context rather than just the previous item [Liepa 1977]. This makes it possible to store sequences with repeated items. To store **abc**, the trace is:

$$\mathbf{a} + \mathbf{a} \circledast \mathbf{b} + \mathbf{a} \circledast \mathbf{b} \circledast \mathbf{c}.$$

This type of sequence can be retrieved in a similar way to the previous, except that the retrieval cue must be built up using convolutions.

The retrieval of later items in both these representations could be improved by subtracting off prefix components as the items in the sequence are retrieved.

Yet another way to represent sequences is to use a fixed cue for each position of the sequence. I call this method for storing sequences *trajectory-association*, because each element of the sequence is associated with a point along a predetermined trajectory. To store **abc**, the trace is:

$$\mathbf{p}_1 \circledast \mathbf{a} + \mathbf{p}_2 \circledast \mathbf{b} + \mathbf{p}_3 \circledast \mathbf{c}.$$

The retrieval (and storage) cues $\mathbf{p}_i$ can be arbitrary or generated in some manner from a single vector, e.g., $\mathbf{p}_i = (\mathbf{p})^i$, i.e., $\mathbf{p}$ raised to the i th convolution power (Section 3.6.5). Convolution powers provide an easy way of generating a sequence of uncorrelated vectors from a single vector. In Chapter 5 I describe how trajectory-associated sequences can be decoded by a recurrent network. Trajectory-association can also be applied to representing continuous trajectories by using fractional powers of $\mathbf{p}$. Fractional convolution powers are also discussed in Section 3.6.5. The choice of $\mathbf{p}$ is important, as the length of $(\mathbf{p})^i$ can increase exponentially with i. It is best to make $\mathbf{p}$ a unitary vector (Section 3.6.3), because if $\mathbf{p}$ is unitary, $|(\mathbf{p})^i| = 1$ for all i.

Multiple methods can be combined, e.g., we can combine positional cues and Liepa's context method to store **abc** as:

$$\mathbf{p}_1 \circledast \mathbf{a} + (\mathbf{p}_2 + \mathbf{a}) \circledast \mathbf{b} + (\mathbf{p}_3 + \mathbf{a} \circledast \mathbf{b}) \circledast \mathbf{c}.$$

Either cue can be used the decode the sequence, but the reconstruction noise will be lowest when both cues are used together.

These methods for representing sequences can also be used to represent stacks. For example, a stack of n items, $\mathbf{x}_1 \cdots \mathbf{x}_n$, with $\mathbf{x}_1$ on top, can be represented by

$$\mathbf{s} = \mathbf{x}_1 + \mathbf{p} \circledast \mathbf{x}_2 + \mathbf{p} \circledast \mathbf{p} \circledast \mathbf{x}_3 + \cdots + \mathbf{p}^n \circledast \mathbf{x}_n.$$

The functions for manipulating such a stack are as follows:

$$\begin{aligned} \text{push}(\mathbf{s}, \mathbf{x}) &= \mathbf{x} + \mathbf{p} \circledast \mathbf{s} && \text{(function value is the new stack)} \\ \text{top}(\mathbf{s}) &= \text{clean-up}(\mathbf{s}) && \text{(function value is the top item)} \\ \text{pop}(\mathbf{s}) &= (\mathbf{s} - \text{top}(\mathbf{s})) \circledast \mathbf{p}^* && \text{(function value is the new stack)} \end{aligned}$$

An empty stack is noticed when the clean-up operation finds nothing similar to $\mathbf{s}$.

A problem with this type of stack implementation is that $\text{pop}(\text{push}(\mathbf{s}, \mathbf{x})) = \mathbf{s} \circledast \mathbf{p} \circledast \mathbf{p}^*$ is only approximately equal to $\mathbf{s}$. This is because $\mathbf{p}^*$ is an approximate inverse. A consequence

is that successive pushes and pops at one level lead to the continual degradation of the lower-level items. After a pair of push-pop actions, the stack will be $\mathbf{s} \circledast \mathbf{p} \circledast \mathbf{p}^*$, which is only approximately equal to $\mathbf{s}$. Additional push-pop pairs further corrupt the remaining part of the stack. There are two possible solutions to this problem – use chunking (see next section) or restrict $\mathbf{p}$ to be a vector for which the exact inverse is equal to the approximate inverse, in which case $\mathbf{s} \circledast \mathbf{p} \circledast \mathbf{p}^* = \mathbf{s}$ (see Section 3.6.3).

3.4.2 Chunking of sequences

All of the above methods have soft limits on the length of sequences that can be stored. As the sequences get longer the noise in the retrieved items increases until the items are impossible to identify. This limit can be overcome by chunking — creating new "non-terminal" items representing subsequences.

The second sequence representation method is more suitable for chunking. Suppose we want to represent the sequence **abcdefgh**. We can create three new items representing subsequences:

$$\begin{aligned} \mathbf{s}_{abc} &= \mathbf{a} + \mathbf{a} \circledast \mathbf{b} + \mathbf{a} \circledast \mathbf{b} \circledast \mathbf{c} \\ \mathbf{s}_{de} &= \mathbf{d} + \mathbf{d} \circledast \mathbf{e} \\ \mathbf{s}_{fgh} &= \mathbf{f} + \mathbf{f} \circledast \mathbf{g} + \mathbf{f} \circledast \mathbf{g} \circledast \mathbf{h} \end{aligned}$$

These new items must be added to the clean-up memory because they are new chunks. The creation of new chunks is what gives this method the ability to represent sequences of arbitrary length. The representation for the whole sequence is:

$$\mathbf{s}_{abc} + \mathbf{s}_{abc} \circledast \mathbf{s}_{de} + \mathbf{s}_{abc} \circledast \mathbf{s}_{de} \circledast \mathbf{s}_{fgh}.$$

Decoding this chunked sequence is slightly more difficult, requiring the use of a stack and decisions on whether an item is a non-terminal that should be further decoded.

3.4.3 Variable binding

It is simple to implement variable binding with convolution: convolve the variable representation with the value representation. For example, the binding of the value **a** to the variable **x** and the value **b** to the variable **y** is

$$\mathbf{t} = \mathbf{x} \circledast \mathbf{a} + \mathbf{y} \circledast \mathbf{b}.$$

Variables can be unbound by convolving the binding with the approximate inverse of the variable. This binding method allows multiple instances of a variable in a trace to be substituted for in a single-operation (approximately). It also allows us to find out what variable is bound to a given value.

3.4.4 Holographic Reduced Representations for simple predicates

Predicates or relations can be represented as simple frame-like structures using convolution encoding, in a manner analogous to the outer products of roles and fillers in Hinton [1981] and Smolensky [1990] and the frames of DUCS [Touretzky and Geva 1987]. A frame consists of a frame name and a set of roles, each represented by a vector. An instantiated frame is the superposition of the frame name and the role-filler bindings (roles convolved

with their respective fillers). This instantiated frame is a reduced representation for the frame, I call it a Holographic Reduced Representation (HRR). For example, suppose we have a (very simplified) frame for eating. The vector for the frame name is **eat** and the vectors for the roles are $\mathbf{eat}_{agt}$ and $\mathbf{eat}_{obj}$. This frame can be instantiated with the fillers **mark** and **the_fish**, to represent "Mark ate the fish":[4]

$$\mathbf{s}_1 = \mathbf{eat} + \mathbf{eat}_{agt} \circledast \mathbf{mark} + \mathbf{eat}_{obj} \circledast \mathbf{the_fish}$$

Fillers (or roles) can be retrieved from the instantiated frame by convolving with the approximate inverse of the role (or filler). The role vectors for different frames can be frame specific, i.e., $\mathbf{eat}_{agt}$ can be different from $\mathbf{see}_{agt}$, or they can be the same (or just similar). The frame name is included as a retrieval cue; it is not necessary for role-filler binding and unbinding. It makes all eat-frames somewhat similar to the **eat** vector (and thus to each other). Eat-frames can be retrieved from clean-up memory by probing with the **eat** vector.

A role-filler binding such as $\mathbf{eat}_{agt} \circledast \mathbf{mark}$ is not similar to either the role or the filler, because the expected value of $\mathbf{x} \circledast \mathbf{y} \cdot \mathbf{x}$ is zero. It is also not similar to a binding of **mark** with a dissimilar role, because the expected value of $\mathbf{x} \circledast \mathbf{y} \cdot \mathbf{x} \circledast \mathbf{z}$ is zero. This means that two different frames with the same fillers can have no similarity. If this is not desired, it is easy to make such frames similar by including the fillers in the frame:

$$\mathbf{s}_1' = \mathbf{eat} + \mathbf{mark} + \mathbf{the_fish} + \mathbf{eat}_{agt} \circledast \mathbf{mark} + \mathbf{eat}_{obj} \circledast \mathbf{the_fish}$$

This makes the representation for a frame somewhat similar to its fillers and to other frames involving the same fillers. It also allows fillers to be used as retrieval cues, because $\mathbf{s}_1'$ is similar to **mark** + **the_fish**. Including fillers in the frame is not without cost – additional vectors in a convolution trace always increase the noise in recognition and reconstruction.

3.4.5 Holographic Reduced Representations for nested predicates

The vector representation of a frame is of the same dimension as the vector representation of a filler and can be used as a filler in another frame. In this way, convolution encoding affords the representation of hierarchical structure in a fixed width vector.

For example, we can use an instantiated frame from the previous section as a filler in another frame representing "Hunger caused Mark to eat the fish":

$$\begin{aligned}\mathbf{s}_2 &= \mathbf{cause} + \mathbf{cause}_{agt} \circledast \mathbf{hunger} + \mathbf{cause}_{obj} \circledast \mathbf{s}_1 \\ &= \mathbf{cause} + \mathbf{cause}_{agt} \circledast \mathbf{hunger} + \mathbf{cause}_{obj} \circledast \mathbf{eat} \\ &\quad + \mathbf{cause}_{obj} \circledast \mathbf{eat}_{agt} \circledast \mathbf{mark} + \mathbf{cause}_{obj} \circledast \mathbf{eat}_{obj} \circledast \mathbf{the_fish}\end{aligned}$$

Normalization of Euclidean lengths of the subframes becomes an issue, because $\mathbf{s}_1$ has a Euclidean length of $\sqrt{3}$ but it probably should be given the same weight as **hunger** in $\mathbf{s}_2$. I discuss the issue of normalization in Section 3.5.3.

This HRR will be similar to other HRRs which have similar role-filler bindings or frame names. This HRR can be made similar to other frames which involve similar subframes or fillers in different roles by adding the filler vectors into the HRR:

$$\mathbf{s}_2' = \mathbf{cause} + \mathbf{mark} + \mathbf{the_fish} + \mathbf{hunger} + \mathbf{s}_1 + \mathbf{cause}_{agt} \circledast \mathbf{hunger} + \mathbf{cause}_{obj} \circledast \mathbf{s}_1$$

[4] I do not attempt to represent tense in any of the examples in this thesis. All the examples are in the past tense, and I use the infinitive forms of verbs for vector names.

It is better not to include fillers in the HRRs for the subframes ($\mathbf{s}_1$ here), because doing so introduces bindings, such as $\mathbf{cause}_{obj} \circledast \mathbf{mark}$, which can be undesirable in some situations.

These recursive representations can be manipulated with or without chunking. Without chunking, we could extract the agent of the object by convolving with $(\mathbf{cause}_{obj} \circledast \mathbf{eat}_{agt})^* = \mathbf{cause}_{obj}{}^* \circledast \mathbf{eat}_{agt}{}^*$. Using chunking, we could first extract the object, clean it up, and then extract its agent, giving a less noisy result. There is a tradeoff between accuracy and speed — if intermediate chunks are not cleaned up the retrievals are faster but less accurate. I discuss the decoding of frames and give results from a simulation in Section 3.10.

The commutativity of the circular convolution operation can cause ambiguity in some situations. This results from the fact that $\mathbf{t} \circledast \mathbf{r}_1^* \circledast \mathbf{r}_2^* = \mathbf{t} \circledast \mathbf{r}_2^* \circledast \mathbf{r}_1^*$. The ambiguity is greatly alleviated by using frame specific role vectors rather than generic role vectors (e.g., a generic "agent" vector.) A situation when ambiguity can still arise is when two instantiations of the same frame are nested in another instantiation of that same frame. In this case the agent of the object can be confused with the object of the agent. Whether this causes problems remains to be seen. In any case, there are variants of circular convolution that are not commutative (Section 3.6.7).

Holographic reduced representations provide a way of realizing of Hinton's [1990] hypothetical system that could, in the same physical set of units, either focus attention on constituents or have the whole meaning present at once. Furthermore, the systematic relationship between the representations for components and frames (i.e., reduced descriptions) means that frames do not need to be decoded to gain some information about the components (see Section 3.10.2).

3.4.6 Equivalence of first-order and higher-order representations for roles

In Section 2.4.3 I reviewed two tensor-product role-filler representations for predicates. Smolensky [1990] suggests that the predicate `p(a,b)` can be represented as a second-order tensor $\mathbf{r}_1 \otimes \mathbf{a} + \mathbf{r}_2 \otimes \mathbf{b}$. Dolan and Smolensky [1989] suggest a third-order tensor: $\mathbf{p} \otimes \mathbf{r}_1 \otimes \mathbf{a} + \mathbf{p} \otimes \mathbf{r}_2 \otimes \mathbf{b}$. The difference between these two representations is that the first uses a first-order tensor for the roles (e.g., $\mathbf{r}_1$), whereas the second uses a second-order tensor for the roles (e.g., $\mathbf{p} \otimes \mathbf{r}_1$). With HRRs, this difference is largely immaterial – $\mathbf{p} \circledast \mathbf{r}_1$ is just a vector, like $\mathbf{r}_1$. In most circumstances the use of these will be equivalent, though it is conceivable that the higher-order role representation could support cleverer ways of decoding HRRs.

The same remarks apply to the frame name. The two styles of representations, exemplified by

$$\mathbf{s}_1 = \mathbf{eat} + \mathbf{eat}_{agt} \circledast \mathbf{mark} + \mathbf{eat}_{obj} \circledast \mathbf{the_fish}$$

and

$$\mathbf{s}_1' = \mathbf{framename} \circledast \mathbf{eat} + \mathbf{eat}_{agt} \circledast \mathbf{mark} + \mathbf{eat}_{obj} \circledast \mathbf{the_fish},$$

where **framename** is common to all frames, have the same characteristics. Different instantiations of the same predicate will be similar, and different predicates will be different. The only difference is in how the frame name can be accessed. The second style of representation ($\mathbf{s}_1'$) allows one to decode the frame name from the HRR with $\mathbf{framename}^*$, giving the frame name superimposed with noise, which is easy to clean up. The first style of representation ($\mathbf{s}_1$) has the frame name superimposed with the role filler bindings, which can be more difficult to clean up, unless frame names are kept in a separate clean-up memory.

3.4.7 What can we do with reduced representations?

There are two ways we can use reduced representations of sequences and predicates. One way is to decode them in order to reconstruct full representations, which involves either reading out a sequence, or finding the fillers of roles. If accurate results are required, the decoded vectors must be cleaned up, which involves more computation. The other way to use HRRs is without decoding, for doing such things as computing the similarity between HRRs. Similarity can be used as the criterion for retrieval from clean-up memory (long-term memory), if the clean-up memory is content-addressable. If we want to use undecoded HRRs to compute the similarity of items, then we must make sure that the dot-product of HRRs is an adequate measure of similarity. This is the justification for adding vectors such as fillers and predicate names into HRRs. In Chapter 6 I describe how to incorporate additional bindings which improve the degree to which the similarity of HRRs reflects the analogical similarity of structures. However, we must be judicious in adding more vectors to HRRs, because each additional vector creates more noise in both decoding and similarity judgements.

3.5 Constraints on the vectors and the representation of features and tokens

In many connectionist systems, the vectors representing items are analyzed in terms of "microfeatures". For example, Hinton's "Family trees" network [Hinton 1990] learned microfeatures representing concepts such as age and nationality (Section 2.2.2). The requirement of HRRs that elements of vectors be randomly and independently distributed seems at odds with this interpretation. Furthermore, if every element of a vector is regarded as a "microfeature" it is unclear how to use the large number of them that the vectors of HRRs provide.

3.5.1 The representation of features, types, and tokens

There is no requirement that single features be represented by single bits in a distributed representation. Features also can be represented by high-dimensional distributed representations as wide as the representation of the whole object. An item having some features can be partly the sum of those features. Tokens of a type can be distinguished from each by the superposition of some identity-giving vector that is unique for each token. Features can be represented by random vectors. For example, the person "Mark" can be represented by $\mathbf{mark} = \mathbf{being} + \mathbf{human} + \mathbf{id}_{mark}$, where $\mathbf{id}_{mark}$ is some random vector that distinguishes **mark** from representations of other people. Each component feature can be weighted according to its importance or salience, if necessary.

This scheme has the following advantages over a local microfeature representation:

- The representation of any feature of an item will degrade gracefully as the elements of the vector representing the item are corrupted.
- The number of features in an item is only loosely related to the dimensionality of the vectors representing items.
- The vectors can be of as high a dimension as desired, and higher dimensionality will give better fidelity in the representation of features.

- The vectors representing items can be expressed as sums of vectors with random independently distributed elements.

When a set of vectors representing items is constructed from distributed features in this way the elements of the vectors will not be consistent with being drawn from independent distributions. However, if linear superposition and circular convolution are used to construct representations, all the expressions describing the recall and matching of vectors can be expanded in terms of the random feature vectors. Thus the means and variances for the signals in a system with non-random vectors, and consequently the probabilities of correct retrieval, can be analytically derived. This allows analysis of the crosstalk induced by having similar vectors representing similar entities (e.g., $\mathbf{mark} = \mathbf{being} + \mathbf{human} + \mathbf{id}_{mark}$, and $\mathbf{john} = \mathbf{being} + \mathbf{human} + \mathbf{id}_{john}$). This analysis is done in Sections 3.10.4 and 3.10.5.

The idea of distributing features over the entire vector representing an item is not new. It is a linear transform and has been suggested by other authors under the name "randomization" or "random maps" (e.g., Schönemann [1987]).

Care must be taken that the "ownership" of features is not confused when using this method to represent features (or attributes) of objects. Ambiguity of feature ownership can arise when multiple objects are stored in a superposition memory. For example, suppose colour and shape are encoded as additive components. If the representations for "red circle" and "blue triangle" were summed, the result would be the same as for the sum of "red triangle" and "blue circle". However, note that if the representations were convolved with distinct vectors (e.g., different role vectors) before they were added the results would not be ambiguous:

$$(\mathbf{red} + \mathbf{triangle}) \circledast \mathbf{role}_1 + (\mathbf{blue} + \mathbf{circle}) \circledast \mathbf{role}_2$$

is distinct from

$$(\mathbf{red} + \mathbf{circle}) \circledast \mathbf{role}_1 + (\mathbf{blue} + \mathbf{triangle}) \circledast \mathbf{role}_2.$$

In any case, it sometimes may be necessary to convolve features with objects to avoid ambiguity, e.g., to use $\mathbf{red} + \mathbf{circle} + \mathbf{red} \circledast \mathbf{circle}$ for a red circle, and $\mathbf{human} + \mathbf{id}_{john} + \mathbf{human} \circledast \mathbf{id}_{john}$ for John. The disadvantages of doing this are lower similarity of an entity and its features (e.g., red-circle and red), lower similarity of similar entities (e.g., red-circle and red-square) and slightly greater complexity. I did not find it necessary to do this in any of the examples in this thesis.

3.5.2 Why binary distributed representations are difficult to use

Binary distributed representations have some advantages over distributed representations in $\mathcal{R}^n$. They use fewer bits to store the same amount of information (in terms of numbers of items), and can be used with non-linear memories (e.g., Willshaw *et al*'s [1969] non-linear correlograph), which are more information-efficient and give reconstructions with higher signal-to-noise ratios.

However, there is one problem with using binary representations in systems in which the representations of some items are the superpositions of the representation of other items. The problem is that it is difficult to maintain constant density in representations, which is essential for the correct functioning of memories like the non-linear correlograph. For example, how are the two vectors (01001100) and (00100101) to be superimposed to give a binary vector with density 3/8? This may not be an insurmountable problem, but I

do not know any good solutions. Hence, in this thesis I use distributed representations in $\mathcal{R}^n$.

3.5.3 Normalization of vectors

The magnitude (Euclidean length) of all vectors (HRRs, tokens, and base vectors) should be equal, or close to, 1. There are two reasons for this. The first is that we want the dot-product to reflect similarity rather than vector magnitude. The second is that when we superimpose two vectors, we want each to be equally represented in the result (we can use scalar weights if we want one vector to be more prominent in the sum).

There are two ways to make the magnitude of vectors close to 1: *normalization* and *scaling by expected magnitude*. Normalization makes the magnitude of vectors exactly 1, and careful scaling by expected magnitude the expected magnitude of vectors equal to 1.

The normalized version of the vector $\mathbf{x}$ is denoted by $\langle \mathbf{x} \rangle$ and is defined as follows (the denominator is the Euclidean length of $\mathbf{x}$):

$$\langle \mathbf{x} \rangle = \frac{\mathbf{x}}{\sqrt{\sum_{i=0}^{n-1} x_i^2}}$$

The expected magnitude of a vector can be made equal to 1 by scaling:

$$\mathbf{x}' = \frac{\mathbf{x}}{\mathrm{E}[|\mathbf{x}|]}$$

This is differs from normalization in two ways. The magnitude of the scaled vector is not guaranteed to be equal to 1, but it has an expected value of 1. Also, only information about the distributions of the x_i and not their actual values, enters into the computation of the scaling factor.

New vectors can be created in three different ways: as a random base vector, as the convolution product of several vectors, or as the superposition of several vectors. Superposition is the main concern, because the magnitude of a superposition of several vectors is nearly always greater than the magnitudes of the individual vectors.

The expected magnitude of vectors with 512 elements chosen independently from $\mathrm{N}(0, 1/512)$ (i.e., base vectors) is 1. For high-dimensional vectors the variance is small, e.g., with 512-dimensional vectors, and standard deviation of the vector magnitude will be $\sqrt{(2/512)} = 0.0625$.

The expected magnitude of a convolution product is equal to the product of the magnitudes, provided that the vectors do not have common components (i.e., their expected dot-product is zero). If two vectors have expected magnitudes of 1, then their convolution product will have an expected magnitude of 1. The expected magnitude of an auto-convolution product, e.g., $\mathbf{a} \circledast \mathbf{a}$, is equal to $\sqrt{2}$ if the a_i are independently distributed as $\mathrm{N}(0, 1/n)$ (see Table 3.1 in Section 3.6.1). For this reason, I avoid using the auto-convolution product.

The expected magnitude of a superposition of vectors is equal to the square root of the sum of the squares of the magnitudes, again provided that the vectors do not have common components. If we have a superposition of M independently chosen vectors, then we scale by $\sqrt{M}$ to make the expected magnitude of the superposition equal to 1. For example, we scale $\mathbf{a} + \mathbf{b}$ by $\sqrt{2}$. The only way to scale correctly when vectors are not chosen independently is to decompose the vectors to a set of independently chosen ones. For example, suppose $\mathbf{x} = (\mathbf{a} + \mathbf{b})/\sqrt{2}$ and $\mathbf{y} = (\mathbf{a} + \mathbf{c})/\sqrt{2}$ (where $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ are independently chosen). The correct way to scale $\mathbf{z} = \mathbf{x} + \mathbf{y}$ is by expressing it as $\mathbf{z} = 2\mathbf{a}/\sqrt{2} + \mathbf{b}/\sqrt{2} + \mathbf{c}/\sqrt{2}$: the scaling factor is $\sqrt{3}$.

In general, normalization is both simpler to use and produces more reliable results, because it reduces the variance of dot-products (see Appendix G). However, using normalization invalidates the simple capacity analyses presented in this thesis, so in cases where I analyze capacity I use scaling. These capacity analyses provide a lower bound – performance improves if normalization is substituted for scaling.

3.5.4 Are the constraints on vectors reasonable?

Fisher, Lippincott and Lee [1987] point out that the constraints on vectors which holographic memories impose make them difficult to use in practice. This is because most vectors produced by sensory apparatus or other components of a physical system are unlikely to satisfy these constraints. Other types of associative memory can handle a wider range of vectors (and provide reconstructions with higher signal-to-noise ratios). Some researchers[5] conclude from this that holographic memories are probably not worthy of further study.

However, this argument is made only in the context of considering the properties of different schemes for storing associations between pairs of items. HRRs do more than this. Other types of associative memories have many problems when applied to representing complex structure, and thus do not provide a clearly superior alternative as they do in the case of storing pairwise associations.

If it is desired to interface a system which uses HRRs with another system that uses vector representations which do not conform to the constraints (e.g., a perceptual system), a different associative memory can be used to translate between representations. The combination of a convolution-based memory (for HRRs) and another associative memory (for mapping between non-conforming and conforming representations) allows the representation of complex associations that are difficult to represent without using a convolution-based memory.

3.6 Mathematical Properties

Circular convolution may be regarded as a multiplication operation over vectors: two vectors multiplied together (convolved) result in another vector. A finite-dimensional vector space over the real numbers, with circular convolution as multiplication and the usual definitions of scalar multiplication and vector addition, forms a commutative linear algebra. This is most easily proved using the observation that convolution corresponds to elementwise multiplication in a different basis, as described in Section 3.6.2. All the rules that apply to scalar algebra (i.e., associativity and commutativity of addition and multiplication, and distributivity of multiplication over addition) also apply to this algebra. This makes it very easy to manipulate expressions containing additions, convolutions, and scalar multiplications.

This algebra has many of the same properties as the algebra considered by Borsellino and Poggio [1973] and Schönemann [1987], which had unwrapped convolution as a multiplication operation over an infinite dimensional vector space restricted to vectors with

[5] E.g., an anonymous reviewer of Plate [in press].

	Expression	mean	variance
(1)	$\mathbf{a} \cdot \mathbf{a}$	1	$\frac{2}{n}$
(2)	$\mathbf{a} \cdot \mathbf{b}$	0	$\frac{1}{n}$
(3)	$\mathbf{a} \cdot \mathbf{a} \circledast \mathbf{b}$	0	$\frac{2n+1}{n^2}$
(4)	$\mathbf{a} \cdot \mathbf{b} \circledast \mathbf{c}$	0	$\frac{1}{n}$
(5)	$\mathbf{a} \circledast \mathbf{a} \cdot \mathbf{a} \circledast \mathbf{a} = \mathbf{a} \circledast \mathbf{a} \circledast \mathbf{a}^* \cdot \mathbf{a}$	$2 + \frac{2}{n}$	$\frac{40n+112}{n^2}$
(6)	$\mathbf{a} \circledast \mathbf{b} \cdot \mathbf{a} \circledast \mathbf{b} = \mathbf{a} \circledast \mathbf{b} \circledast \mathbf{a}^* \cdot \mathbf{b}$	1	$\frac{6n+4}{n^2}$
(7)	$\mathbf{a} \circledast \mathbf{b} \cdot \mathbf{a} \circledast \mathbf{a} = \mathbf{a} \circledast \mathbf{b} \circledast \mathbf{a}^* \cdot \mathbf{a}$	0	$\frac{6n+18}{n^2}$
(8)	$\mathbf{a} \circledast \mathbf{b} \cdot \mathbf{a} \circledast \mathbf{c} = \mathbf{a} \circledast \mathbf{b} \circledast \mathbf{a}^* \cdot \mathbf{c}$	0	$\frac{2n+2}{n^2}$
(9)	$\mathbf{a} \circledast \mathbf{b} \cdot \mathbf{c} \circledast \mathbf{c} = \mathbf{a} \circledast \mathbf{b} \circledast \mathbf{c}^* \cdot \mathbf{c}$	0	$\frac{2n+2}{n^2}$
(10)	$\mathbf{a} \circledast \mathbf{b} \cdot \mathbf{c} \circledast \mathbf{d} = \mathbf{a} \circledast \mathbf{b} \circledast \mathbf{c}^* \cdot \mathbf{d}$	0	$\frac{1}{n}$

Table 3.1: Means and variances of dot-products of common convolution expressions for vectors with elements distributed as $N(0, 1/n)$, where n is the dimensionality of the vectors. These expressions assume no normalization or scaling. The dot-products are normally distributed in the limit as n goes to infinity.

a finite number of non-zero elements. Schönemann observed that representing the correlation of **b** and **a** as a convolution of **a** with an involution of **b** made expressions with convolutions and correlations easier to manipulate.

3.6.1 Distributions of dot-products

The distributions of the dot-products of vectors and convolutions of vectors can be analytically derived. I have calculated distributions for some dot-products; these are shown in Table 3.1.[6] The calculations are based on the assumption that the elements for the vectors **a**, **b**, **c**, and **d** are independently distributed as $N(0, 1/n)$ (the expected length of these vectors is 1). The central limit theorem tells us that these dot-products are normally distributed in the limit as n goes to infinity because they are the sum of products of individual vector elements.[7] The equivalent expressions in rows (5) to (10) are derived from the following identity of convolution algebra:

$$\mathbf{a} \circledast \mathbf{b} \circledast \mathbf{x}^* \cdot \mathbf{y} \;=\; \mathbf{a} \circledast \mathbf{b} \circledast \mathbf{y}^* \cdot \mathbf{x} \;=\; \mathbf{a} \circledast \mathbf{b} \cdot \mathbf{x} \circledast \mathbf{y}$$

The variances for decoding circular-convolution bindings are slightly higher than the variances for decoding unwrapped-convolution bindings (see Murdock [1985] for a table similar to Table 3.1). However, the difference is only a small constant factor (1.3 to 1.5), and is probably due to to the fact that circular-convolution bindings are represented over n elements rather than the $2n - 1$ elements for wrapped-convolution bindings.

[6] These were not calculated analytically. The numbers of variance and covariance terms are described by quadratic polynomials, so I worked out exact expressions for low values of n (4,6,8,10,12 and 16) and fitted polynomials to these (such as $2n^2 + n$ for $\mathbf{a} \cdot \mathbf{a} \circledast \mathbf{b}$). I also checked that the expressions agreed with means and variances from numerical simulations for high values of n (1024, 2048 and 4096).

[7] Although there are some correlations among these products there is sufficient independence for the central limit theorem to apply for even quite small n.

These means and variances are used as follows: Suppose that **a**, **b**, **c**, **d**, and **e** are random vectors with elements drawn independently from $N(0, 1/n)$. Then the value of $\mathbf{a}^* \circledast (\mathbf{a} \circledast \mathbf{b} + \mathbf{c} \circledast \mathbf{d}) \cdot \mathbf{b}$ will have an expected value of 1 and a variance of $\frac{7n+4}{n^2}$ ($=\frac{6n+4}{n^2} + \frac{1}{n}$ using rows 6 and 10 in Table 3.1). The value of $\mathbf{a}^* \circledast (\mathbf{a} \circledast \mathbf{b} + \mathbf{c} \circledast \mathbf{d}) \cdot \mathbf{e}$ will have an expected value of 0 and a variance of $\frac{3n+2}{n^2}$ ($= \frac{2n+2}{n^2} + \frac{1}{n}$ using rows 8 and 10 in Table 3.1).

Of some interest is the distribution of the elements of $\mathbf{a} \circledast \mathbf{b}$. If the elements of **a** and **b** are independently distributed as $N(0, 1/n)$ then the mean of the elements of $\mathbf{a} \circledast \mathbf{b}$ is 0 but the variance is higher than $1/n$ and the covariance of the elements is not zero. The expected length of $\mathbf{a} \circledast \mathbf{b}$ is still 1, provided that the elements of **a** are distributed independently of those of **b** (the expected length of $\mathbf{a} \circledast \mathbf{a}$ is $\sqrt{2 + 2/n}$). Thus, the variance of $\mathbf{a} \circledast \mathbf{b} \circledast \mathbf{c} \circledast \mathbf{a}^* \circledast \mathbf{b}^* \cdot \mathbf{c}$ is higher than that of $\mathbf{a} \circledast \mathbf{b} \circledast \mathbf{a}^* \cdot \mathbf{b}$. A consequence of this is that some care must be taken when using $\mathbf{a} \circledast \mathbf{b}$ as a storage cue, especially in the case where $\mathbf{a} = \mathbf{b}$. This is particularly relevant to the storage capabilities of HRRs because when recursive frames are stored, convolution products e.g., $\mathbf{cause}_{obj} \circledast \mathbf{eat}_{agt}$, are the storage cues.

These means and variances are different if normalization is used. In Appendix G, I report experiments which investigate this. The results show that variances are significantly lower and more uniform when normalization is used. In Chapter 4, I describe a different type of normalization which results in even lower and more uniform variances.

3.6.2 Using FFTs to compute convolution

The fastest way to compute convolution is via Fast Fourier transforms (FFTs) [Brigham 1974]. The computation involves a transformation, an elementwise multiplication of two vectors, and an inverse transformation. Computing convolution via these three steps takes $O(n \log n)$, whereas the obvious implementation of the convolution equation $c_i = \sum_j a_j b_{i-j}$ takes $O(n^2)$ time.[8]

The discrete Fourier transform, $\mathbf{f} : \mathcal{C}^n \rightarrow \mathcal{C}^n$, ($\mathcal{C}$ is the field of complex numbers) is defined as:

$$f_j(\mathbf{x}) = \sum_{k=0}^{n-1} x_k e^{-i2\pi jk/n},$$

where $i^2 = -1$ and $f_j(\mathbf{x})$ is the jth element of $\mathbf{f}(\mathbf{x})$. The discrete Fourier transform is invertible and defines a one-to-one relationship between vectors in the spatial and frequency domains. It can be computed in $O(n \log n)$ time using the Fast Fourier Transform (FFT) algorithm. The inverse discrete Fourier transform is similar:

$$f_j^{-1}(\mathbf{x}) = \frac{1}{n} \sum_{k=0}^{n-1} x_k e^{i2\pi jk/n}$$

and can also be computed in $O(n \log n)$ time using the FFT algorithm.

The equation relating convolution and the Fourier transform is:

$$\mathbf{x} \circledast \mathbf{y} = \mathbf{f}'(\mathbf{f}(\mathbf{x}) \odot \mathbf{f}(\mathbf{y})),$$

where $\odot$ is the elementwise multiplication of two vectors. Figure 3.5 illustrates the elementwise multiplication of the transforms of two vectors **x** and **y**. The constraints on the Fourier transforms of real vectors are apparent in this figure: the angles of $f_0(\mathbf{x})$ and $f_{n/2}(\mathbf{x})$ are zero, and $f_{i+n/2}(\mathbf{x})$ is the complex conjugate of $f_i(\mathbf{x})$ (for $i \neq 0$).[9]

[8] Computing convolution via FFTs takes about the same time as the $O(n^2)$ method for $n = 32$. It is faster for $n > 32$ and slower for $n < 32$.

[9] If a vector in the frequency domain does not satisfy these constraints, then it is the Fourier transform of

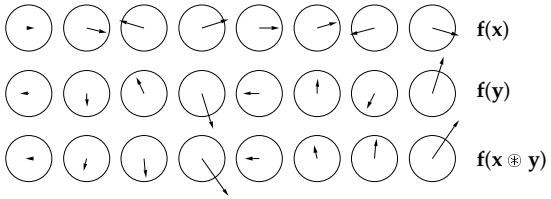


Figure 3.5: The convolution of 8-dimensional vectors **x** and **y** in the frequency domain. Each arrow represents a complex number in a unit circle. The angle of a component of $\mathbf{f}(\mathbf{x} \circledast \mathbf{y})$ is equal to the sum of the angles of the corresponding components of $\mathbf{f}(\mathbf{x})$ and $\mathbf{f}(\mathbf{y})$. The magnitude of a component of $\mathbf{f}(\mathbf{x} \circledast \mathbf{y})$ is equal to the product of the magnitudes of the corresponding components of $\mathbf{f}(\mathbf{x})$ and $\mathbf{f}(\mathbf{y})$.

In the following sections, I shall refer to the original space as the *spatial domain*, and the range of Fourier transform as the *frequency domain*. Both domains are n-dimensional vector spaces, and both the forward and inverse Fourier transforms are linear. In Chapter 4, I describe how all the operations required for HRRs can be performed solely in the frequency domain.

3.6.3 Approximate and exact inverses in the frequency domain

Since convolution in the spatial domain is equivalent to elementwise multiplication in the frequency domain we can easily find convolutive inverses in the frequency domain. By definition, **y** is the inverse of **x** if $\mathbf{x} \circledast \mathbf{y} = \mathbf{1}$ and we can write $\mathbf{y} = \mathbf{x}^{-1}$. The convolutive identity vector is $\mathbf{1} = (1, 0, \ldots, 0)$. Transforming this into the frequency domain gives

$$\mathbf{f}(\mathbf{x}) \odot \mathbf{f}(\mathbf{x}^{-1}) = \mathbf{f}(\mathbf{1}).$$

The transform of the identity is

$$\mathbf{f}(\mathbf{1}) = (e^{0i}, e^{0i}, \ldots, e^{0i}) = (1, 1, \ldots, 1).$$

This gives independent relationships between the corresponding elements of $\mathbf{f}(\mathbf{x})$ and $\mathbf{f}(\mathbf{x}*)$ which can be expressed as

$$f_j(\mathbf{x}^{-1}) f_j(\mathbf{x}) = 1.$$

Expressing $\mathbf{f}(\mathbf{x})$ in polar coordinates gives

$$f_j(\mathbf{x}) = r_j e^{i\theta_j}$$

and we can see that the Fourier transform of the inverse of **x** must be:

$$f_j(\mathbf{x}^{-1}) = \frac{1}{r_j} e^{-i\theta_j}.$$

Now consider the approximate inverse. It can be seen from the definition of the Fourier transform that the transform of the involution of **x** is

$$f_j(\mathbf{x}^*) = r_j e^{-i\theta_j}.$$

a complex vector. Put another way, if $\mathbf{X} \in \mathcal{C}^n$ (the frequency domain) does not satisfy these constraints, then $f^{-1}(\mathbf{X})$ exists, but is a complex-valued vector.

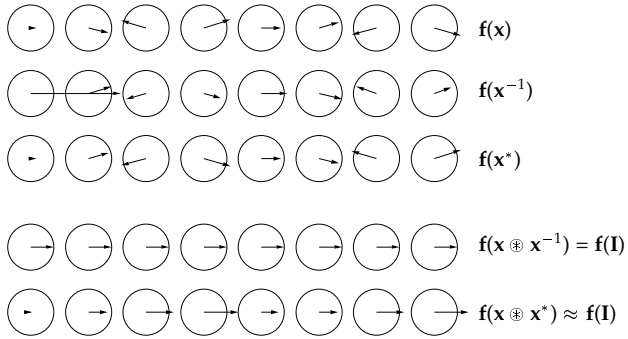


Figure 3.6: The exact and approximate inverses of a vector in the frequency domain ($\mathbf{f}(\mathbf{x}^{-1})$ and $\mathbf{f}(\mathbf{x}^*)$ respectively. $\mathbf{f}(\mathbf{x} \circledast \mathbf{x}^{-1})$ is the identity vector, and $\mathbf{f}(\mathbf{x} \circledast \mathbf{x}^*)$ is an approximation to it.

In the frequency domain, the elements of the approximate inverse have the same magnitudes as the original elements, whereas the elements of the exact inverse have magnitudes equal to the reciprocals of the original elements. It follows that the approximate inverse is equal to the exact inverse if and only if the magnitude of all frequency components is 1 (i.e., $r_j = 1$ for all j). I refer to this class of vectors as *unitary* vectors.[10] Put another way, **x** is a unitary vector if and only if $\mathbf{x} \circledast \mathbf{x}^* = \mathbf{I}$ (i.e., $\mathbf{x} \circledast \mathbf{x} = \mathbf{I}$). Figure 3.6 shows the approximate and exact inverses of a vector.

Vectors which have zero magnitudes for one or more frequency components have no exact inverse, so I refer to these as *singular* vectors.

Unitary vectors and vectors with elements distributed as $N(0, 1/n)$ in the spatial domain are closely related. The mean of frequency component magnitudes for the latter is 1 (except for the two components with zero angle), and the phase angles are uniformly distributed. Transformed into the spatial domain, the elements of unitary vectors are distributed as $N(1/n, 1/n)$. The elements are not independent – there are only $(n/2 - 1)$ degrees of freedom.

3.6.4 Why the exact inverse is not always useful

Since $\mathbf{x}^{-1}$ can be used to decode $\mathbf{x} \circledast \mathbf{y}$ exactly, it might seem to be a better candidate for the decoding vector than the approximate inverse $\mathbf{x}^*$. However, using the exact inverse results in a lower signal-to-noise ratio in the retrieved vector when the memory trace is noisy or when there are other vectors added into it.[11] This problem arises because the exact inverse is unstable. For vectors with elements independently distributed as $N(0, 1/n)$, $|\mathbf{x}^{-1}|$ is often significantly greater than $|\mathbf{x}|$ (except when **x** is unitary). However, $|\mathbf{x}^*|$ is always equal to $|\mathbf{x}|$. The reason for this can be understood by considering the inverses in the frequency domain. The relationship between the vector magnitude and the magnitudes of the frequency components is given by Parseval's theorem:

$$|\mathbf{x}|^2 = \sum_k x_k^2 = \frac{1}{n} \sum_k |f_k(\mathbf{x})|^2.$$

[10] By analogy to unitary matrices, for which the conjugate transpose is equal to the inverse.

[11] Inverse filters are well-known to be sensitive to noise [Elliot 1986].

Taking the approximate inverse does not change the magnitude of the frequency components, so $|\mathbf{x}^*|$ is always equal to $|\mathbf{x}|$. However, taking the exact inverse involves the reciprocals of the magnitudes, so if the magnitude of a frequency component of $\mathbf{x}$ is close to zero (which can happen), the magnitude of the corresponding frequency component of $\mathbf{x}^{-1}$ will be very large magnitude. If exact inverses are used for decoding a noisy trace, the high-magnitude frequency components magnify the noise in the trace without contributing much signal to the reconstruction.

3.6.5 The convolutive power of a vector

The convolutive power of a vector (exponentiation) can be straightforwardly defined by exponentiation of its elements in the frequency domain, i.e.,

$$f_j(\mathbf{x}^k) = (f_j(\mathbf{x}))^k.$$

It is easy to verify that this definition satisfies $\mathbf{x}^0 = \bar{I}$, and $\mathbf{x}^i \circledast \mathbf{x} = \mathbf{x}^{i+1}$ for $i \geq 0$. For negative and fractional exponents powers are most easily expressed using polar coordinates. Let $f_j(\mathbf{x}) = r_j e^{i\theta_j}$, where $r_j \leq 0$ and $-\pi < \theta \leq \pi$. Then

$$f_j(\mathbf{x}^k) = r_j^k e^{ik\theta_j}.$$

Note that for any vector $\mathbf{x} \in \mathcal{R}^n$, and $\mathbf{k} \in \mathcal{R}$, $\mathbf{x}^k$ is also in $\mathcal{R}^n$, because exponentiation as defined above does not affect the conditions which make a vector in the frequency domain the transform of a real vector in the spatial domain (see Section 3.6.2). (This would not be true if in the above defintion we used $\leq 0\theta < 2\pi$, or allowed r to be negative.) Also, unitary vectors are closed under exponentiation: any power of a unitary vector is a unitary vector.

Integer powers are useful for generating some types of encoding keys (cf. Section 3.4.1) and fractional powers can be used to represent trajectories through continuous space, as done in Chapter 5.

3.6.6 Matrices corresponding to circular convolution

The convolution operation can be expressed as a matrix-vector multiplication.

$$\mathbf{a} \circledast \mathbf{b} = M_a \mathbf{b}$$

where M_a is the matrix corresponding to convolution by $\mathbf{a}$. It has elements $m_{a_{ij}} = a_{i-j}$ (where the subscripts on $\mathbf{a}$ are interpreted modulo n). Such matrices are known as "circulant matrices" [Davis 1979]. Since the mapping computed by the connections between two layers in a feedforward network is a matrix-vector multiplication, it is possible for such a mapping to correspond to convolution by a fixed vector.

The algebra carries through with matrix multiplication: $M_{a \circledast b} = M_a M_b$. Transposition corresponds to the approximate inverse, i.e., $M_{a^*} = M_a^T$. Also, if $\mathbf{a}$ is a unitary vector, then M_a is a real unitary matrix (i.e., an orthonormal matrix). The eigenvalues of M_a are the individual (complex valued) elements of the Fourier transform of $\mathbf{a}$. The corresponding eigenvectors are the inverse transforms of the frequency components, i.e., $[1, 0, 0, \ldots]$, $[0, 1, 0, \ldots]$, etc, in the frequency domain.

3.6.7 Non-commutative variants and analogs of convolution.

The commutativity of convolution can cause ambiguity in the representations of some structures. If this is a problem, non-commutative variants of circular convolution can be constructed by permuting the elements of the argument vectors in either the spatial or frequency domain. The permutations applied to right and left vectors must be different. The resulting operation is neither commutative nor associative, but is bilinear (and thus distributes over addition and preserves similarity) and has an easily computed approximate inverse.

An alternative operation that is non-commutative but still associative is matrix multiplication. This could be used to associate two vectors by treating each vector as a square matrix. The dimension of the vectors would have to be a perfect square. I am unaware of what the scaling and interference properties of such an associative memory operation would be. It would be similarity-preserving and vectors corresponding to orthogonal matrices would have simple inverses.

Another possibility is to use a random convolution-like compression of the outer product, as mentioned in Section 7.4.

3.6.8 Partial derivatives of convolutions

A convolution operation can be used in a feedforward networks[12] and values can be propagated forward in $O(n \log n)$ time (on serial machines). Derivatives can also be backpropagated in $O(n \log n)$ time. Suppose we have a network in which the values from two groups of units are convolved together and sent to a third group of units. The relevant portion of such a feedforward network is shown in Figure 3.7. Suppose we have the partial derivatives $\frac{\partial E}{\partial c_i}$ of outputs of the convolution with respect to an objective function E. Then the partial derivatives of the inputs to the convolution can be calculated as follows:

$$\begin{aligned}
\frac{\partial E}{\partial a_k} &= \textstyle\sum_i \frac{\partial E}{\partial c_i} \frac{\partial c_i}{\partial a_k} \\
&= \textstyle\sum_i \frac{\partial E}{\partial c_i} b_{i-k} \\
&= \textstyle\sum_i \frac{\partial E}{\partial c_i} [\mathbf{b}^*]_{k-i} \\
&= [\partial_c \circledast \mathbf{b}^*]_k
\end{aligned}$$

where ∂_c is the vector with elements $\frac{\partial E}{\partial c_i}$, and $[\cdot]_k$ is the kth element of a vector.

This means that it is possible to incorporate a convolution operation in a feedforward network and do the forward and backpropagation computations for the convolution in $O(n \log n)$ time. One reason one might want to do this could be to use a backpropagation network to learn good vector representations for items for some specific task. This is pursued in Chapter 5.

3.7 Faster comparison with items in clean-up memory

Computing the dot-product of the probe with each item in clean-up memory can be expensive if there are a large number of items in clean-up memory, especially on a serial machine.

[12] For an introduction to feedforward networks see [Rumelhart, Hinton and J. 1986].

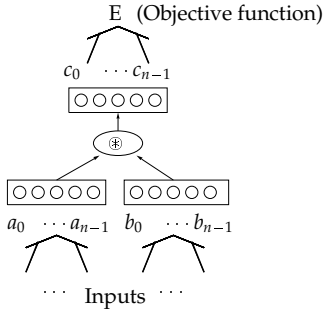


Figure 3.7: A convolution operation in a backpropagation network.

This computation can be reduced by computing a fast estimate of the dot-product and then only computing the floating-point dot-product for the most promising candidates. The fast estimate is the count of matching bits in binary versions of the vectors. To make this practical, we must store a binary version along with each item in clean-up memory. The binary version is a n-bit vector, and the nth bit is 1 if the nth element of the item vector is positive, and 0 otherwise. The dot-product of binary vectors (the bitwise dot-product) can be computed many times faster, using full-word XOR operations, than the dot-product of floating-point vectors. The bitwise dot-product is highly correlated with the floating-point dot-product – for $n = 2048$ the correlation coefficient was measured to be 0.997 (for 4096 pairs of vectors). This of course depends on HRR vector elements having a mean of 0, but this will be true if the HRRs are constructed following the descriptions in this chapter.

Figure 3.8 shows some simulation results. These plots show the bitwise dot-product versus the floating-point dot-product. It is clear that the correlation between the floating-point and the bitwise dot-products increases with vector dimension. This means that the bitwise dot-product is increasingly useful with higher dimension vectors. In Appendix I, I report some simulations in which this technique significantly reduced the amount of computation done by the clean-up operation.

The data in Figure 3.8 was generated by randomly choosing r in $[-1, 1]$, and the vectors $\mathbf{x}$ and $\mathbf{y}$ with elements randomly selected from $N(0, 1/n)$. Two vectors $\mathbf{a}$ and $\mathbf{b}$ were derived from $\mathbf{x}$, $\mathbf{y}$, and r in such a way that the expected dot-product of $\mathbf{a}$ and $\mathbf{b}$ was r (recall that $\langle \mathbf{x} \rangle$ denotes the normalized version of $\mathbf{x}$):

$$\begin{aligned} \mathbf{a} &= \langle \mathbf{x} \rangle \\ \mathbf{b} &= \langle r\mathbf{a} + \sqrt{1 - r^2}\, \mathbf{y} \rangle \end{aligned}$$

The floating-point dot-product $\mathbf{a} \cdot \mathbf{b}$ is plotted against the scaled count of matching bits in the binary versions of $\mathbf{a}$ and $\mathbf{b}$. Each plot contains 4096 data points.

When searching for the vector in clean-up memory which is the closest to some vector $\mathbf{x}$, the following scheme reduces the number of floating-point dot-products that must be computed:

1. Compute and record the bitwise dot-product of $\mathbf{x}$ with all the vectors in clean-up memory.

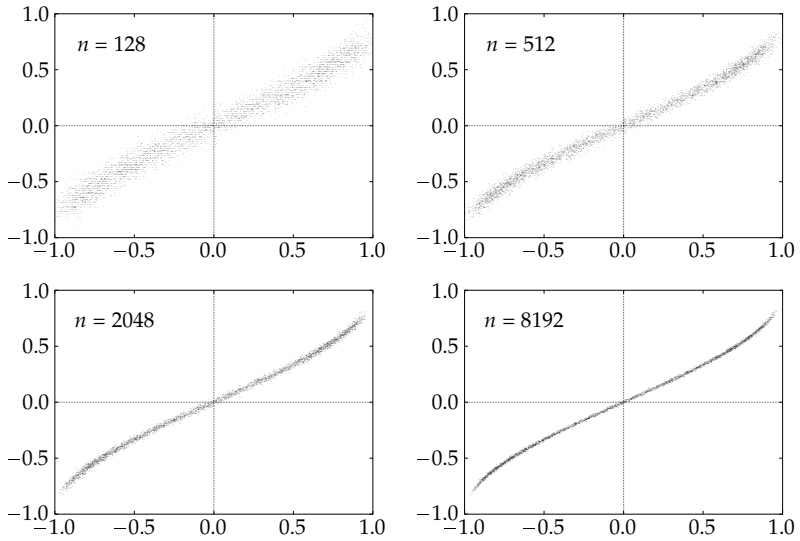


Figure 3.8: Bitwise dot-products (vertical axis) versus floating-point dot-products (horizontal axis) for various vector dimensions.

2. Let the maximum bitwise dot-product be t. Calculate the lower bound on what the floating-point dot-product corresponding to t could be, let this be t'. Calculate the lower bound on the bitwise dot-product of vectors which have a floating-point dot-product of t', let this be t''.

3. Compute and record the floating-point dot-products of $\mathbf{x}$ with all the vectors in clean-up memory which had a bitwise dot-product of t'' or greater.

This scheme saves computation by not computing the floating-point dot-product for vectors which could not possibly be the best match. The lower bounds must be derived from some statistical model of the differences between bitwise and floating-point dot-products. Some preliminary analysis of simulation data (such as is shown in Figure 3.8) indicates that we can use

$$t'' = t - s, \quad \text{where} \quad s = 6e^{-0.505 \log n}.$$

This corresponds to an interval at least 8 standard deviations wide.

3.8 The capacity of convolution memories and HRRs

In Appendices B through E, I investigate the capacities of superposition and paired-associates convolution memory. For superposition and pairwise (variable-binding) convolution memories, I demonstrate the following scaling properties:

- The number of pairs (items) that can be stored in a single trace is approximately linear in n.
- The probability of error is negative exponential in n.
- The number of items that can be stored in memory in clean-up memory is exponential in n.

For example, if we use vectors with 4096 elements, and have 100,000 items in clean-up memory, where 100 of those are similar to each other (with dot-product 0.5), then we can store approximately a dozen bindings in a single trace, and have a 99% chance of decoding the trace correctly (from Figure D.3).

If some items in clean-up memory are similar to each other, as they will if they are tokens or HRRs constructed according the to suggestions in this thesis, then the capacity equation tends to be dominated by the number of similar items, rather than the total number of items in clean-up memory. Thus the simulations described in this thesis, which mostly have a small number of similar items, would still work with a much larger number of items in clean-up memory, provided that no particular item in clean-up memory was similar to many others.

In Appendices D and E I only consider the capacity of paired-associates convolution memory. In order to predict the capacity of a system using HRRs, it is necessary to consider the effect of higher-order bindings. If unnormalized vectors are used, the results of decoding higher-order bindings will be much noisier than the results of decoding pairwise associations, and thus the capacity will be lower. This is because the variance for decoding higher-order convolution products is higher than that for decoding lower-order convolution products. E.g., the variance of $(\mathbf{a} \circledast \mathbf{b} \circledast \mathbf{c}) \circledast (\mathbf{b} \circledast \mathbf{c})^* \cdot \mathbf{a}$ is greater than the variance of $(\mathbf{a} \circledast \mathbf{b}) \circledast \mathbf{b}^* \cdot \mathbf{a}$. If normalized vectors are used, the results will be only slightly noisier, and the capacity slightly lower. If unitary vectors are used, as discussed in Chapter 4, the results will not be any noisier than for pairwise associations.

Thus, if normalization is used, the size of a structure that can be encoded in a HRR, i.e., the number of terms in the expanded convolution expression, can be expected to increase almost linearly with the vector dimension. The results of decoding deep structures will be noiser, but if this becomes a critical problem it can be overcome by storing substructures as separate chunks, and cleaning up intermediate results. In fact, if chunking is used, structures of unlimited size can be stored, as long as all the component chunks are not too large.

3.9 Convolution-based memories versus matrix-based memories

Convolution- and matrix-based memories have long been considered alternative ways of implementing associative distributed memory (e.g., see Willshaw *et al* [1969], Willshaw [1981b], Pike [1984], and Fisher *et al* [1987]). The focus has usually been on storing pairwise associations, though Pike does consider higher-order associations. Convolution-based memories were invented first, inspired by holography [Reichardt 1957; Gabor 1968b; Longuet-Higgins 1968; Willshaw, Buneman and Longuet-Higgins 1969; Borsellino and Poggio 1973]. Matrix memories were proposed by Willshaw, Buneman and Longuet-Higgins [1969] as a way of improving on the properties of convolution-based memories, while retaining their distributed associative character. Since then, convolution-based memories have fallen out of favor. Willshaw [1981a], commenting on his earlier work on convolution-based memory, writes:

> The prominence given to the newly emerging field of holography in the 1960s made it likely that before long the suitability of the hologram as a distributed associative memory would be explored. But as seen from today's viewpoint, a study of the hologram may seem an irrelevant diversion. This was, however, one of the routes to the particular associative memory that we investigated, the well-known matrix, or correlation, memory, to which holographic models bear a formal, mathematical similarity.

Matrix memories have gone on to be widely studied, e.g., in the form of Hopfield nets [1982]. In this thesis I return to convolution-based memories because they are more suitable for forming complex associations, and their weaknesses can be compensated for by using them in conjunctive with associative clean-up memories.

3.9.1 Comparison of convolution and matrix-based memories

Matrix and convolution-based memories have much in common – they share the following properties (where $\mathbf{a}/\mathbf{b}$ means the association of $\mathbf{a}$ and $\mathbf{b}$):

- Learning is a one-shot process – an association can be stored after a single presentation, in contrast to schemes involving iterative feedback.[13]
- The framework involves three operations: a bilinear encoding operation to create associations, superposition to combine associations, and a bilinear decoding operation.
- Incorporating non-linearities in the form of thresholds on the three operations improves performance and capacity [Willshaw 1981b].
- Non-linear versions have the same information efficiency under optimal conditions [Willshaw 1981b], and linear versions also have similar information efficiency [Pike 1984].
- Association preserves similarity: If items $\mathbf{a}$ and $\mathbf{a}'$ are similar, then the traces $\mathbf{a}/\mathbf{b}$ and $\mathbf{a}'/\mathbf{b}$ will also be similar (to approximately the same degree). This is useful because, if the representations of items make similarity explicit (Section 1.4.2), then structures composed of those items will also make similarity explicit.
- Association has randomizing effect: If items $\mathbf{x}$ and $\mathbf{y}$ are not similar, then the traces $\mathbf{x}/\mathbf{a}$ and $\mathbf{y}/\mathbf{a}$ will also not be similar. This is useful because it allows superimposing multiple associations without getting false cross-associates.
- The encoding operation involves pairwise products of the elements of each vector.

The differences between convolution- and matrix-based memories are as follows:

- Convolution-based memories are symmetric, and matrix memories are not. This is because convolution is commutative ($\mathbf{a} \circledast \mathbf{b} = \mathbf{b} \circledast \mathbf{a}$), whereas the outer product is not.

[13] This is not to say that iterative schemes cannot improve performance. On the contrary, in Chapter 5 I describe how such a scheme enables the use of lower-dimensional vectors.

- In matrix memories the dimensionality of the trace is the square of the dimensionality of the vectors being associated, whereas in convolution memories, the dimensionality of the trace can be made the same as that of the vectors being associated.
- Matrix memories can store more associations of vectors of a given dimensionality (though the size of the traces are correspondingly larger).
- Reconstructions from linear matrix memories have higher signal-to-noise ratios than those from linear convolution-based memories.
- It is easier to build error-correcting properties into matrix memories, so that if the association $\mathbf{a}/\mathbf{b}$ is decoded with $\mathbf{a}'$ (a corrupted version of $\mathbf{a}$) the reconstructed version of $\mathbf{b}$ will be more similar to $\mathbf{b}$ than $\mathbf{a}'$ is to $\mathbf{a}$.

3.9.2 Convolution as a low-dimensional projection of a tensor product

The convolution operation can be seen as a realization of Smolensky, Legendre and Miyata's [1992] suggestion for solving the expanding-dimensionality problem of tensor products by projecting higher-order tensor products onto a lower-dimensional space. The projection matrix P such that $P\,\mathbf{x} \otimes \mathbf{y} = \mathbf{x} \circledast \mathbf{y}$ is easily derived. The 3×9 matrix P which projects a 9 element tensor onto to three elements is shown below. It has a regular structure: each 3×3 block contains one wrapped diagonal.

$$P\,\mathbf{x} \otimes \mathbf{y} = \begin{bmatrix} 1&0&0&0&0&1&0&1&0 \\ 0&1&0&1&0&0&0&0&1 \\ 0&0&1&0&1&0&1&0&0 \end{bmatrix} \begin{bmatrix} x_1y_1 \\ x_1y_2 \\ x_1y_3 \\ x_2y_1 \\ x_2y_2 \\ x_2y_3 \\ x_3y_1 \\ x_3y_2 \\ x_3y_3 \end{bmatrix} = \begin{bmatrix} x_1y_1 + x_3y_2 + x_2y_3 \\ x_2y_1 + x_1y_2 + x_3y_3 \\ x_3y_1 + x_2y_2 + x_1y_3 \end{bmatrix} = \mathbf{x} \circledast \mathbf{y}$$

In general, the dimensionality-reducing projection P has a null space which interferes minimally with associations. For any non-singular vector $\mathbf{x}$, there are no distinct vectors $\mathbf{y}$ and $\mathbf{y}'$ such that $P\,\mathbf{x} \otimes \mathbf{y} = P\,\mathbf{x} \otimes \mathbf{y}'$ (i.e., such that $\mathbf{x} \circledast \mathbf{y} = \mathbf{x} \circledast \mathbf{y}'$). It is true that there are an infinite number of distinct pairs $\mathbf{x}, \mathbf{y}$ and $\mathbf{x}', \mathbf{y}'$ such that $P\,\mathbf{x} \otimes \mathbf{y} = P\,\mathbf{x}' \otimes \mathbf{y}'$, but this does not cause problems if either (a) we know one of the items in the association, or (b) we choose vectors randomly so that this type of collision has low probability.

3.10 An example of encoding and decoding HRRs

An example of HRR frame construction and decoding is presented in this section. The types and tokens representing objects and concepts are constructed according to the suggestions in Section 3.5. Results from a simulation of the example using 512-dimensional vectors are reported.

Object features		Role features	Frame names
being	**food**	**object**	**cause**
human	**fish**	**agent**	**eat**
state	**bread**		**see**

Table 3.2: Base vectors. Vector elements are all independently chosen from $N(0, 1/512)$.

$$\begin{aligned}
\textbf{mark} &= (\textbf{being} + \textbf{human} + \textbf{id}_{mark})/\sqrt{3} \\
\textbf{john} &= (\textbf{being} + \textbf{human} + \textbf{id}_{john})/\sqrt{3} \\
\textbf{paul} &= (\textbf{being} + \textbf{human} + \textbf{id}_{paul})/\sqrt{3} \\
\textbf{luke} &= (\textbf{being} + \textbf{human} + \textbf{id}_{luke})/\sqrt{3} \\
\textbf{the_fish} &= (\textbf{food} + \textbf{fish} + \textbf{id}_{the_fish})/\sqrt{3} \\
\textbf{the_bread} &= (\textbf{food} + \textbf{bread} + \textbf{id}_{the_bread})/\sqrt{3} \\
\textbf{hunger} &= (\textbf{state} + \textbf{id}_{hunger})/\sqrt{2} \\
\textbf{thirst} &= (\textbf{state} + \textbf{id}_{thirst})/\sqrt{2} \\
\textbf{eat}_{agt} &= (\textbf{agent} + \textbf{id}_{eat_agent})/\sqrt{2} \\
\textbf{eat}_{obj} &= (\textbf{object} + \textbf{id}_{eat_object})/\sqrt{2}
\end{aligned}$$

Table 3.3: Token and role vectors constructed from base vectors and random identity-giving vectors. The identity vectors (e.g., $\textbf{id}_{mark}$) are chosen in the same way as the base vectors. The denominators are chosen so that the expected length of a vector is 1.0. Other roles (e.g., $\textbf{see}_{agt}$) are constructed in the analogous fashion.

3.10.1 Representation and similarity of tokens

The suggestion in Section 3.5 for token vectors (representing an instance of a type) was that they be composed of the sum of features and a distinguishing vector giving individual identity. In this example the base vectors (the features other vectors are composed of) have elements chosen independently from $N(0, 1/512)$. The base vectors are listed in Table 3.2. The token and role vectors are constructed by summing the relevant feature vectors and a distinguishing random "identity" vector that is used to give a distinct identity to an instance of a type. Scale factors are included in order to make the expected length of the vectors equal to 1. These token vectors and a representative pair of role vectors are listed in Table 3.3.

The similarity matrix of the tokens is shown in Table 3.4. Tokens with more features in common have higher similarity (e.g., **mark** and **john**), and tokens with no features in common have very low similarity (e.g., **john** and **the_fish**).

I weighted vectors to make their expected length equal to 1 rather than normalizing them because normalization invalidates the analysis of expectations and variances of dot-products. In practice, it is better to normalize vectors.

	mark	**john**	**paul**	**luke**	**the_fish**	**the_bread**	**hunger**	**thirst**
mark	1.07							
john	0.78	1.08						
paul	0.76	0.75	1.08					
luke	0.73	0.68	0.74	1.01				
the_fish	0.01	0.00	-.02	-.03	1.16			
the_bread	0.02	0.01	0.06	0.01	0.35	0.97		
hunger	0.01	0.06	0.05	0.03	0.10	0.03	0.93	
thirst	0.01	0.11	0.04	0.06	0.07	0.02	0.48	1.04

Table 3.4: Similarities (dot-products) among some of the tokens. The diagonal elements are the squares of the vector lengths. Tokens sharing feature vectors (see Table 3.3) have higher similarity.

$\mathbf{s}_1$	Mark ate the fish.
$\mathbf{s}_2$	Hunger caused Mark to eat the fish.
$\mathbf{s}_3$	John ate.
$\mathbf{s}_4$	John saw Mark.
$\mathbf{s}_5$	John saw the fish.
$\mathbf{s}_6$	The fish saw John.

Table 3.5: Sentences.

3.10.2 Representation and similarity of frames

The six sentences listed in Table 3.5 are represented as HRR frames. The expressions for these HRRs are listed in Table 3.6. Again, scale factors are included to make the expected length of the vectors equal to 1.

The similarities of the HRRs are shown in Table 3.7. Some similarities between instantiated frames can be detected without decoding. The HRRs for similar sentences ($\mathbf{s}_4$, $\mathbf{s}_5$, and $\mathbf{s}_6$) are similar, because convolution preserves similarity. One role-filler binding is similar to another to the extent that their respective roles and fillers are similar. This is investigated in more detail in Section 3.10.5.

The HRRs for two frames which have identical constituents but different structure are distinct. Consequently, $\mathbf{s}_5$ and $\mathbf{s}_6$ are distinct. Having the same filler in the same role causes more similarity between frames than having the same filler in a different role. Consequently, $\mathbf{s}_4$ is more similar to $\mathbf{s}_5$ than $\mathbf{s}_6$, because **john** fills the agent role in $\mathbf{s}_4$ and $\mathbf{s}_5$, and the object roles in $\mathbf{s}_6$.

3.10.3 Extracting fillers and roles from frames

The filler of a particular role in a frame is extracted as follows: the frame is convolved with the approximate inverse of the role and the result is cleaned up by choosing the most

$$
\begin{aligned}
\mathbf{s}_1 &= (\mathbf{eat} + \mathbf{eat}_{agt} \circledast \mathbf{mark} + \mathbf{eat}_{obj} \circledast \mathbf{the_fish})/\sqrt{3} \\
\mathbf{s}_2 &= (\mathbf{cause} + \mathbf{cause}_{agt} \circledast \mathbf{hunger} + \mathbf{cause}_{obj} \circledast \mathbf{s}_1)/\sqrt{3} \\
\mathbf{s}_3 &= (\mathbf{eat} + \mathbf{eat}_{agt} \circledast \mathbf{john})/\sqrt{2} \\
\mathbf{s}_4 &= (\mathbf{see} + \mathbf{see}_{agt} \circledast \mathbf{john} + \mathbf{see}_{obj} \circledast \mathbf{mark})/\sqrt{3} \\
\mathbf{s}_5 &= (\mathbf{see} + \mathbf{see}_{agt} \circledast \mathbf{john} + \mathbf{see}_{obj} \circledast \mathbf{the_fish})/\sqrt{3} \\
\mathbf{s}_6 &= (\mathbf{see} + \mathbf{see}_{agt} \circledast \mathbf{the_fish} + \mathbf{see}_{obj} \circledast \mathbf{john})/\sqrt{3}
\end{aligned}
$$

Table 3.6: HRR frame vectors representing the sentences in Table 3.5

	$\mathbf{s}_1$	$\mathbf{s}_2$	$\mathbf{s}_3$	$\mathbf{s}_4$	$\mathbf{s}_5$	$\mathbf{s}_6$
$\mathbf{s}_1$	1.14					
$\mathbf{s}_2$	0.02	0.98				
$\mathbf{s}_3$	0.81	0.01	1.11			
$\mathbf{s}_4$	0.11	0.12	0.25	1.13		
$\mathbf{s}_5$	0.30	0.05	0.31	0.73	0.99	
$\mathbf{s}_6$	-.01	0.14	0.01	0.65	0.35	1.14

Table 3.7: Similarities (dot-products) among the frames.

similar vector in the clean-up memory. The clean-up memory contains all feature, token, role, and frame vectors (i.e., all the vectors listed in Tables 3.2, 3.3, and 3.6).

The extraction of various fillers and roles is shown in Table 3.8. For each extraction, the three vectors in clean-up memory that are most similar to the result are shown. In all cases the most similar object is the correct one.

As shown in Row (1), the expression to extract the agent of $\mathbf{s}_1$ is

$$x = \mathbf{s}_1 \circledast \mathbf{eat}_{agt}{}^{*}.$$

The three objects in clean-up memory most similar to x (with their respective dot-products) are **mark** (0.62), **john** (0.47), and **paul** (0.41). The filler of the agent role in $\mathbf{s}_1$ is indeed **mark**, so the extraction has been performed correctly.

The construction of $\mathbf{s}_1$ and the determination of the filler of its object role, on row (1) in Table 3.8, is illustrated in Figure 3.9. In order to enable the perception of similarities among vectors in this figure, the 512-element vectors were laid out in rectangles with dimensions permuted (on all vectors simultaneously) so as to reduce the total sum of variance between neighboring elements. This was done using a simulated annealing program. The reader should not take the visual similarities of the vectors too seriously – dot-product similarity is what is important and is very difficult to judge from merely looking at figures like this.

Row (2) illustrates that the agent of $\mathbf{s}_1$ can also be extracted using the generic agent role (**agent**) rather than the agent role specific to the eat frame ($\mathbf{eat}_{agt}$). The results are stronger when the specific agent is used.

Object to extract	Expression	Similarity scores (dot-product)		
(1) Agent of $\mathbf{s}_1$	$\mathbf{s}_1 \circledast \mathbf{eat}_{agt}^*$	**mark** (0.62)	**john** (0.47)	**paul** (0.41)
(2) Agent of $\mathbf{s}_1$	$\mathbf{s}_1 \circledast \mathbf{agent}^*$	**mark** (0.40)	**john** (0.34)	**human** (0.30)
(3) Object of $\mathbf{s}_1$	$\mathbf{s}_1 \circledast \mathbf{love}_{obj}^*$	**the_fish** (0.69)	**fish** (0.44)	**food** (0.39)
(4) Agent of $\mathbf{s}_2$	$\mathbf{s}_2 \circledast \mathbf{cause}_{agt}^*$	**hunger** (0.50)	**state** (0.39)	**thirst** (0.31)
(5) Object of $\mathbf{s}_2$	$\mathbf{s}_2 \circledast \mathbf{cause}_{obj}^*$	$\mathbf{s}_1$ (0.63)	$\mathbf{s}_3$ (0.46)	**eat** (0.43)
(6) Agent of object of $\mathbf{s}_2$	$\mathbf{s}_2 \circledast \mathbf{cause}_{obj}^* \circledast \mathbf{eat}_{agt}^*$	**mark** (0.27)	**paul** (0.23)	**luke** (0.22)
(7) Object of object of $\mathbf{s}_2$	$\mathbf{s}_2 \circledast \mathbf{cause}_{obj}^* \circledast \mathbf{eat}_{obj}^*$	**the_fish** (0.39)	**fish** (0.24)	**food** (0.23)
(8) Object of $\mathbf{s}_3$	$\mathbf{s}_3 \circledast \mathbf{eat}_{obj}^*$	**food** (0.07)	**the_bread** (0.06)	**mark** (0.06)
(9) John's role in $\mathbf{s}_4$	$\mathbf{s}_4 \circledast \mathbf{john}^*$	$\mathbf{see}_{agt}$ (0.66)	**agent** (0.50)	$\mathbf{see}_{obj}$ (0.47)
(10)John's role in $\mathbf{s}_5$	$\mathbf{s}_5 \circledast \mathbf{john}^*$	$\mathbf{see}_{agt}$ (0.58)	**agent** (0.41)	$\mathbf{eat}_{agt}$ (0.36)

Table 3.8: Results of extracting fillers from the frames. In all cases shown the item most similar to the result is the correct one. The similarity comparisons are all with the entire set of features, tokens, roles, and frames. See the text for discussion of each row.

In $\mathbf{s}_2$ the object role is filled by another frame. There are two alternative methods for extracting the components of this subframe. The first method, which is slower, is to clean up the subframe in clean-up memory (row 5) and then extract its components, as in rows (1) to (3). The second (faster) method, is to omit the clean-up operation and directly convolve the result with the approximate inverses of the roles of $\mathbf{s}_1$. The expressions for the fast method are shown in rows (6) and (7). The first method is an example of using chunking to clean up intermediate results, and gives stronger results at the expense of introducing intermediate clean-up operations. With the intermediate clean-up omitted the chances of error are higher; in row (6) the correct vector is only very slightly stronger than an incorrect one. However, the high-scoring incorrect responses are similar to the correct response; it is clear that the subframe object role filler is a person.

Row (8) shows what happens when we try to extract the filler of an absent role. The frame $\mathbf{s}_3$ ("John ate.") has no object. As expected, $\mathbf{s}_3 \circledast \mathbf{eat}_{obj}^*$ is not significantly similar to anything. Although **food** might seem an appropriate guess, all the responses are weak and it is just a coincidence that **food** gives the strongest response.

It is possible to determine which role a token is filling, as in rows (9) and (10). In $\mathbf{s}_4$, on row (9), the correct role for **john** is $\mathbf{eat}_{agt}$, but $\mathbf{eat}_{obj}$ also scores quite highly. This is because **john** is a person, and a person is also filling the object role in $\mathbf{s}_4$. Compare this with $\mathbf{s}_5$, where the object-role filler (**the_fish**) is not at all similar to the agent-role filler (**john**). The extracted role for **john** is not at all similar to the object role, as shown on row (10).

3.10.4 Probabilities of correct decoding

The expectation and variances of the dot-products

$$\mathbf{s}_1 \circledast \mathbf{eat}_{agt}^* \cdot \mathbf{mark} \quad \text{and} \quad \mathbf{s}_1 \circledast \mathbf{eat}_{agt}^* \cdot \mathbf{p}$$

are calculated in this section (where $\mathbf{p}$ is a vector for a person that is not **mark**). This allows us to calculate the probability that the agent of $\mathbf{s}_1$ will be extracted correctly, as in row (1) of Table 3.8. It must be emphasized that the behaviour of any particular system (i.e., set of vectors) is deterministic. A particular frame in a particular system always will or always will not be decoded correctly. The probabilities calculated in this section are the probabilities that a randomly chosen system will behave in a particular way.

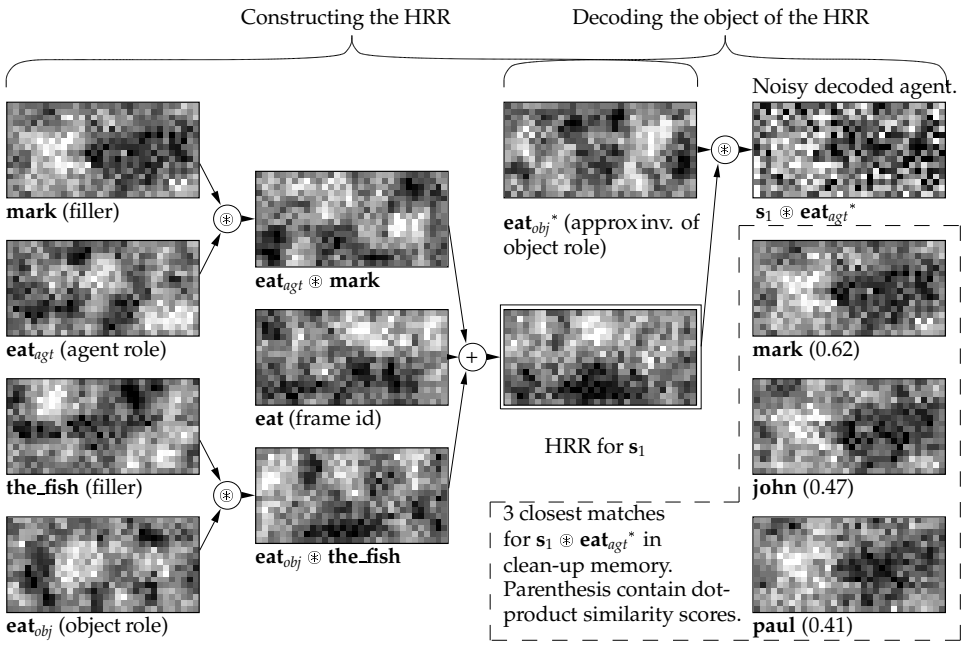


Figure 3.9: Construction and decoding of a HRR for the sentence "Mark ate the fish." ($\mathbf{s}_1$ in Section 3.10). The instantiated frame, labelled $\mathbf{s}_1$, is the sum of role-filler bindings and a frame id (shown in the second column). It is the same dimensionality as all other objects and may be used a filler in another frame (e.g., as in $\mathbf{s}_2$ in Section 3.10). A filler of the HRR can be extracted by convolving the HRR with the approximate inverse of its role. The extraction of the agent-role filler of this sentence is shown on the right (also see Table 3.8). Of the items in clean-up memory, the actual filler, **mark**, is the most similar (shown in the dotted region). The next two most similar items are also shown, with the dot-product match value in parenthesis. The dot-products of the decoded agent with **john** and **paul** might seem high, but these values are expected because the expected similarity of **mark** to other people is 2/3 (and $2/3 \times 0.62 \approx 0.41$). In this high-dimensional space these differences in similarity (0.62 to 0.47) are significant. See Section 3.10.3 for discussion.

Let $\mathbf{d} = \mathbf{s}_1 \circledast \mathbf{eat}_{agt}^*$, then

$$\mathbf{d} \cdot \mathbf{mark} = \mathbf{d} \cdot (\mathbf{being} + \mathbf{human} + \mathbf{id}_{mark})/\sqrt{3}, \textit{ and}$$

$$\mathbf{d} \cdot \mathbf{p} = \mathbf{d} \cdot (\mathbf{being} + \mathbf{human} + \mathbf{id}_{p})/\sqrt{3}.$$

The vector $\mathbf{p}$ is used here as a generic "incorrect person" filler. The extraction is judged to have been performed correctly if $\mathbf{d} \cdot \mathbf{mark} > \mathbf{d} \cdot \mathbf{p}$ for all vectors representing people in the clean-up memory. We can limit the consideration to people-vectors in clean-up memory, because it is extremely unlikely that other vectors will be more similar to $\mathbf{d}$ than the vectors representing people.

It is important to note that these two dot-products are correlated because they share the common term $d \cdot (\mathbf{being} + \mathbf{human})/\sqrt{3}$. To calculate the probabilities accurately it is necessary to take into account the value of this term when choosing the threshold. Let

$$\begin{aligned} X_{mark} &= d \cdot \mathbf{id}_{mark}/\sqrt{3}, \\ X_p &= d \cdot \mathbf{id}_p/\sqrt{3}, \text{ and} \\ Z &= d \cdot (\mathbf{being} + \mathbf{human})/\sqrt{3}. \end{aligned}$$

X_{mark}, X_p, and Z can be regarded as uncorrelated, normally distributed variables, which are derived from the random vectors.

The calculation of the means and variances of X_{mark}, X_p, and Z is presented in Appendix F. For $n = 512$, they are as follows:

	mean	variance	std dev
X_{mark}	0.192	0.00116	0.0341
X_p	0	0.000867	0.0294
Z	0.385	0.00246	0.0496

A lower estimate [14] for the probability P that $Z + X_{mark} > Z + X_p$ for all p is given by

$$P' = \Pr(Z + X_{mark} > t) \cdot \Pr(Z + X_p < t \;\forall\; p \neq \mathbf{mark}),$$

where t is a threshold chosen to maximize this probability.[15]

In this example there are three other people, so

$$P' = \Pr(X_{mark} + Z > t) \cdot \Pr(X_p + Z < t)^3.$$

This has a maximum value of 0.996 for $t = Z + 0.0955$. Thus the probability of correctly identifying the filler of agent role as **mark** in $\mathbf{s}_1$ is at least 0.996. If there were 100 vectors representing people in the clean-up memory, the probability of correctly identifying the decoded agent as **mark** would drop to 0.984.

The reason for calculating means and variances of signals is to estimate the minimum vector dimension that will result in an acceptable probability of error. It is not necessary to calculate the means and variances of all signals, it is sufficient to consider just those pairs whose means are close and whose variances are large.

3.10.5 Similarity-preserving properties of convolution

Convolution is similarity-preserving – two convolution bindings are similar to the degree that their components are similar. For example, the two role bindings $\mathbf{eat}_{agt} \circledast \mathbf{mark}$ and $\mathbf{eat}_{agt} \circledast \mathbf{john}$ are similar because **mark** and **john** are similar.

The bilinearity and randomizing properties of convolution make it possible to calculate analytically the expectation and variance of dot-products of convolution sums and products. The calculation is simple, if somewhat laborious. We use several facts (from Table 3.1) about the dot-products of convolutions of vectors with elements chosen randomly from $N(0, 1/n)$:

[14] $P' \leq P$ because it can be the case that $Z + X_{mark} < t$ and $Z + X_{mark} > Z + X_p \;\forall\; p \neq \mathbf{mark}$.

[15] t is chosen with knowledge of Z but not of X_{mark} or X_p.

Compared with $\mathbf{mark} \circledast \mathbf{eat}_{agt}$		Similarity of filler: 1		2/3		0	
Similarity of role	1	$\mathbf{mark} \circledast \mathbf{eat}_{agt}$	1	$\mathbf{john} \circledast \mathbf{eat}_{agt}$	2/3	$\mathbf{the_fish} \circledast \mathbf{eat}_{agt}$	0
	1/2	$\mathbf{mark} \circledast \mathbf{see}_{agt}$	1/2	$\mathbf{john} \circledast \mathbf{see}_{agt}$	1/3	$\mathbf{the_fish} \circledast \mathbf{see}_{agt}$	0
	0	$\mathbf{mark} \circledast \mathbf{eat}_{obj}$	0	$\mathbf{john} \circledast \mathbf{eat}_{obj}$	0	$\mathbf{the_fish} \circledast \mathbf{eat}_{obj}$	0

Table 3.9: Expected similarity of $\mathbf{mark} \circledast \mathbf{eat}_{agt}$ to other role-filler bindings. One binding is similar to another to the degree that its components are similar.

$$\begin{aligned} \mathrm{E}[(\mathbf{a} \circledast \mathbf{b}) \cdot (\mathbf{a} \circledast \mathbf{b})] &= 1 \\ \mathrm{E}[(\mathbf{a} \circledast \mathbf{b}) \cdot (\mathbf{a} \circledast \mathbf{c})] &= 0 \end{aligned}$$

Let $\mathbf{B}_1$ and $\mathbf{B}_4$ be two role-filler bindings involving similar entities:

$$\begin{aligned} \mathbf{B}_1 &= \mathbf{eat}_{agt} \circledast \mathbf{mark} \\ \mathbf{B}_4 &= \mathbf{eat}_{agt} \circledast \mathbf{john} \\ \mathbf{mark} &= (\mathbf{being} + \mathbf{human} + \mathbf{id}_{mark})/\sqrt{3} \\ \mathbf{john} &= (\mathbf{being} + \mathbf{human} + \mathbf{id}_{john})/\sqrt{3} \end{aligned}$$

where **being**, **human**, $\mathbf{eat}_{agt}$, $\mathbf{id}_{mark}$, and $\mathbf{id}_{john}$ are random base vectors with elements from $N(0, 1/n)$. The scaling factor $1/\sqrt{3}$ makes the expected lengths of **mark** and **john** equal to 1. The vectors **mark** and **john** are similar – their expected dot-product is 2/3. The arithmetic is simpler if we replace **being** + **human** by $\sqrt{2}$**person**, where **person** is a random base vector.

The expected dot-product of $\mathbf{B}_1$ and $\mathbf{B}_4$ is:

$$\begin{aligned} &\mathrm{E}[\mathbf{eat}_{agt} \circledast \mathbf{mark}] \cdot (\mathbf{eat}_{agt} \circledast \mathbf{john}) \\ &= \mathrm{E}[(\mathbf{eat}_{agt} \circledast (\mathbf{person} + \mathbf{id}_{mark})/\sqrt{3}) \cdot (\mathbf{eat}_{agt} \circledast (\mathbf{person} + \mathbf{id}_{john})/\sqrt{3})] \\ &= 2/3\mathrm{E}[\mathbf{eat}_{agt} \circledast \mathbf{person} \cdot \mathbf{eat}_{agt} \circledast \mathbf{person}] + \sqrt{2}/3\mathrm{E}[\mathbf{eat}_{agt} \circledast \mathbf{person} \cdot \mathbf{eat}_{agt} \circledast \mathbf{id}_{john}] \\ &\quad + \sqrt{2}/3\mathrm{E}[\mathbf{eat}_{agt} \circledast \mathbf{id}_{mark} \cdot \mathbf{eat}_{agt} \circledast \mathbf{person}] + 1/3\mathrm{E}[\mathbf{eat}_{agt} \circledast \mathbf{id}_{mark} \cdot \mathbf{eat}_{agt} \circledast \mathbf{id}_{john}] \\ &= 2/3 + 0 + 0 + 0 = 2/3 \end{aligned}$$

In general, the expected similarity of a role-filler binding is equal to the product of the expected similarities of its components.[16] The similarity of a binding to others with varying role and filler similarity is shown in Table 3.9.

So far I have only mentioned the expected dot-product of two vectors, whereas what is usually important is the actual dot-product. The dot-product is a fixed value for a particular pair of vectors. However, we can usefully treat it as a random value that depends on the choice of random base vectors. The dot-products of convolution products and sums are

[16] Care must be taken that one base vector does not appear in both components of a binding, because $\mathrm{E}[\mathbf{a} \circledast \mathbf{a} \cdot \mathbf{a} \circledast \mathbf{a}] = 2 + 2/n$.

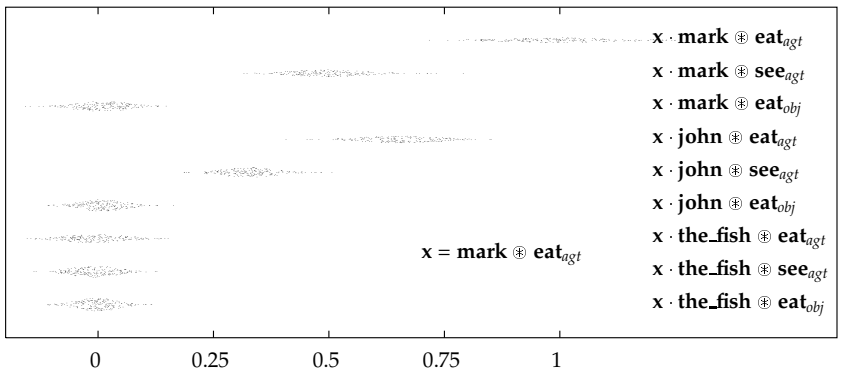


Figure 3.10: Similarities of the role-filler bindings in Table 3.9, from 200 choices of random base vectors, with dimension 512. There are 200 points in each cluster. The variances of the points in each cluster are inversely proportional to the vector dimension. Density-dependent random vertical displacements were added to make the density of points clear. Although the clusters overlap, the order within a single run is generally the same.

normally distributed, and their variance is inversely proportional to n. The observed values of the dot-products in Table 3.9, for 200 choices of random base vectors of dimension 512, are shown in Figure 3.10. Although some of the clusters overlap, e.g., $\mathbf{x} \cdot \mathbf{mark} \circledast \mathbf{see}_{agt}$ and $\mathbf{x} \cdot \mathbf{john} \circledast \mathbf{eat}_{agt}$ (the ones with means at 0.5 and 0.66), these dot-products are rarely out of order in a single run (114 times in 10,000 runs). The reason is that they are correlated – when one is high, the other tends to be high.

In Appendix A, I derive expressions for the means and variances of the similarities shown in Figure 3.10, and compare them to the observed means and variances. They match closely.

3.11 Discussion

3.11.1 Learning and HRRs

One of the attractions of distributed representations is that they can be used in neural networks and trained by example. Hinton's "Family trees" network [1986] demonstrated how a neural network could simultaneously learn a task and the representations for objects in the task. In Chapter 5 I will describe how a recurrent neural can implement the trajectory-association method for storing sequences. The network learns item representations which are superior to randomly chosen ones.

3.11.2 HRRs and Hinton's reduced descriptions

HRRs are not an exact implementation of Hinton's [1990] reduced descriptions (Section 1.5, but they do capture their spirit, and satisfy the four desiderata for reduced descriptions (adequacy, reduction, systematicity, and informativeness about components). A HRR is a reduced description for a predicate – the full representation is the set of fillers and predicate name (and roles). I say that HRRs are not an exact implementation for two reasons: roles are not fixed (as is implied in Figure 1.1), and the full representation need not be laid out in parallel on several groups of units – it can composed and decomposed iteratively.

3.11.3 Chunking

The idea of chunking, or breaking large structures in pieces of manageable size, fits together very well with HRRs. Chunking makes it possible to use HRRs to store structures of unlimited size, and HRRs provide a way of doing chunking that has some very attractive properties.

As more items and bindings are stored in a single HRR the noise on extracted items increases. If too many associations are stored, the quality will be so low that the extracted items will be easily confused with similar items or, in extreme cases, completely unrecognizable. The number of items and bindings in a HRR grows with both the height and width of the structure being represented – $\mathbf{s}_1$ has 3, while $\mathbf{s}_2$ has 5. The number of items and bindings that can be stored for a given degree of quality of decoding grows linearly with the vector dimension (Section 3.8), but using vectors with very high dimension is not a satisfactory way to store large structures. A far superior way is to use chunk information. This involves storing substructures in the clean-up memory, and using them when decoding components of complex structures. For example, to decode the agent of the cause antecedent of $\mathbf{s}_2$ we first extract the cause antecedent. This gives a noisy version of $\mathbf{s}_1$, so we then clean it up by accessing clean-up memory and retrieving the closest match (which should be $\mathbf{s}_1$). Now we have an accurate version of $\mathbf{s}_1$ from which we can extract the filler of the agent role. Using chunks during decoding like this reduces the effective number of items and bindings in HRRs and thus makes it possible to store structures of nearly unlimited size (though of course the number of chunks in clean-up memory grows with the size of the structure).

Any system that uses chunks must have also have a way of referring, or pointing to the chunks. HRRs provide a very attractive way of constructing chunks because a HRR is a pointer that tells us something about what it refers to. This is what the "reduced" in "Holographic Reduced Representation" refers to – a HRR is a compressed, or reduced, version of the structure it represents. The advantage of having a pointer which encodes some information about what it refers to is that some operations can be performed without following the pointer. This can save much time. For example, we can decode nested fillers quickly if very noisy results are acceptable, or we can get an estimate of the similarity of two structures without decoding the structures. In Chapter 6, I investigate how HRRs can be used to compute fast estimates of analogical similarity.

3.11.4 Comparison to other distributed representations for predicates

HRRs have much in common with Smolensky's [1990] tensor-product role-filler representation for predicates. The advantage HRRs have over tensor products is that the dimensions

of HRRs do not expand when they are composed recursively. The natural way of constructing reduced descriptions and chunks in HRRs follows from this. The advantage tensor products have over HRRs is lower noise (zero under some conditions) in decoding. With tensor products, one can do such things as rebind multiple variables in parallel. One can also do this with HRRs, but the noise is considerable.

HRRs differ from Murdock's [1983] and Halford *et al*'s [to appear] predicate representations in that HRRs do not require that any ordering be imposed on the roles, and in the ease of coping with missing fillers. Some common types of decoding, such as finding the filler of a given role, appear to be easier with HRRs. Furthermore, HRRs seem to provide a more natural similarity structure for predicates (see Chapter 6). The type of access that Halford *et al* claim is impossible with role-filler representations (finding one component of a predicate when provided with all the others) can actually be done quite easily with HRRs if a clean-up memory is used. I give an example of this in Appendix I. HRRs are actually more versatile for this type of retrieval because a HRR can be retrieved with fewer than all-but-one components.

Compared to Pollack's [1990] RAAMs, HRRs have the following computational advantages:

- No lengthy training is required for HRRs.
- The retrieval properties of HRRs are simple and predictable, and amenable to analysis.
- Arbitrary structures of arbitrary vectors can be stored in HRRs. The only structures and vectors that one can be confident a RAAM will encode are those it was trained (or tested) on.
- HRRs shed information about nested components, whereas with deeper structures, RAAMs must pack more and more information into the vector. Not all the information about the items in a complex association is stored in the HRR memory trace. Information storage is shared between the convolution memory trace and the associative clean-up memory.
- Many types of structures can be stored in HRRs. HRRs are not constrained to representing trees of a particular degree.
- HRRs have an infinite variety of roles, whereas RAAMs are limited to a small fixed number of positional roles.
- HRRs degrade gracefully in the presence of noise. HRRs normally work with a lot of noise. The effect of adding more noise can be quantified and compensated for by using higher-dimensional vectors. The effect of adding noise to RAAMs is difficult to quantify because they involve nonlinear transformations. Also, RAAMs are unlikely to work well in the presence of noise because they require values to be represented with high precision.
- HRRs do not depend upon representing floating-point values with high precision. Rounding to lower-precision values has much the same effect as adding noise.
- Construction and decomposition of HRRs can be done in $O(n \log n)$ time, using FFTs, whereas RAAMs take $O(n^2)$ time. However, this advantage is probably offset by the high vector dimensions that HRRs require.

HRRs have several disadvantages compared to RAAMs:

- The association operation in HRRs cannot take advantage of the possibility that some structures and items are commonly represented and others rarely. In RAAMs, the association operation is tuned to efficiently represent common structures and items.
- HRRs require very high dimensional vectors.
- For the statistical analysis of HRRs to be valid the elements of the basic vectors must be randomly independently distributed.

3.12 Conclusion

Memory models using circular convolution provide a way of representing compositional structure in distributed representations. They implement Hinton's [1990] suggestion that reduced descriptions should have microfeatures that are systematically derived from those of their constituents. HRRs preserve natural similarity structure – similar frames are represented by similar HRRs. The operations involved are mostly linear and the properties of the scheme are relatively easy to analyze, especially compared to schemes such Pollack's [1990] RAAMs. No learning is required and the scheme works with a wide range of vectors. Systems employing HRRs must have an error-correcting auto-associative memory to clean up the noisy results produced by convolution decoding. Convolution-based memories have low capacity compared to matrix memories, but the storage capacity is sufficient to be usable and scales almost linearly with vector dimension.

HRRs are intended more as an illustration of how convolution and superposition can be used to represent complex structure than as a strict specification of how complex structure should be represented. The essence of HRRs is as follows:

- It is a distributed representation.
- They represent predicate-like objects as the superposition of role-filler bindings (and possibly some additional information).
- The representation of a composition object has the same dimension as the representation of each of its components.
- Roles and fillers are bound together with associative memory operation, like circular convolution, which does not increase dimensionality and is randomizing and similarity preserving.

In any particular domain or application, choices must be made about what will and will not be included in the representation, and about how the representation will be structured. For example, in the HRRs I use in Chapter 6 I superimpose fillers and further bindings derived from the context of fillers. In other domains, fewer or different HRR components might be appropriate.

Chapter 4

HRRs in the frequency domain

The fact that convolution is equivalent to elementwise complex multiplication in the frequency domain suggests that it might be possible to work with complex numbers and avoid convolution and Fourier transforms altogether. Indeed, this is the case, and there are two advantages to doing so. The first is that binding (i.e., convolution) can be computed in $O(n)$ time. The second is that it is simple to force all vectors to be unitary, i.e., to force all frequency components to have a magnitude of 1. This is a particularly effective type of normalization: it makes it possible to use the exact inverse, it normalizes the Euclidean length of vectors, it reduces noise in decoding and comparison, and it results in simpler analytic expressions for the variance of dot-products of bindings. The only difficulty with working with unitary vectors is that there is no operation which corresponds exactly to superposition: I define a substitute operation in this Chapter.

4.1 Circular vectors

In order to work with unitary vectors in the frequency domain, the major representational change required is the use of distributed representations over complex numbers. I define an n-dimensional *circular* vector as a vector of n angles, each between $-\pi$ and π. Each angle can be viewed as the phase of a frequency component. The magnitude of each frequency component is 1, which means that circular vectors are unitary. I use the bar symbol to denote a vector of angles: $\bar{\phi} = [\phi_1, \ldots, \phi_n]^T$. In a *random circular vector*, each of the angles is independently distributed as $U(-\pi, \pi)$, i.e., the uniform distribution over $(-\pi, \pi]$.

The reader may recall from Section 3.6.2 that there are several constraints on the components of the Fourier transform of real-valued vectors: the angles of $f_0(\mathbf{x})$ and $f_{n/2}(\mathbf{x})$ are zero, and for $i \neq 0$, $f_{i+n/2}(\mathbf{x})$ is the complex conjugate of $f_i(\mathbf{x})$. It is unnecessary to maintain these constraints when working with circular vectors, since we can work entirely in the frequency domain. Thus, each angle in a circular vector can independently take on any value in $(-\pi, \pi]$.

4.2 HRR operations with circular vectors

HRRs require three operations: comparison, association (and decoding), and superposition. With the standard representation (real-valued vectors with elements distributed as $N(0, 1/n)$), we use the dot-product, convolution and addition. Using straightforward correspondences between operations in the frequency and spatial domains of the Fourier transform, we can derive comparison and association operations for circular vectors. Superposition is a little more tricky, because in general the sum of unit complex values does not lie on the unit circle.

As explained in Section 3.6.2, convolution in the spatial domain is equivalent to elementwise multiplication in the frequency domains. When all values in the frequency domain lie on the unit circle, this is equivalent to modulo-2π addition of angles. I use $\odot$ to denote this operation.

The similarity of two circular vectors can be measured by the sum of the cosines of the differences between corresponding angles:

$$\bar{\psi} \cdot \bar{\phi} = \frac{1}{n} \sum_j \cos(\psi_i - \phi_i)$$

This exactly corresponds to the dot-product in the spatial domain, because of the identity

$$\mathbf{x} \cdot \mathbf{y} = \frac{1}{n} \sum_j \cos(\theta_j(\mathbf{x}) - \theta_j(\mathbf{y})),$$

where $\theta_j(\mathbf{x})$ is the phase angle of the jth component of the discrete Fourier transform of $\mathbf{x}$. This identity is easily proved by substituting $(\mathbf{z} - \mathbf{y})$ for $\mathbf{x}$ in the discrete version of Parseval's theorem, which states that $\sum_k x_k^2 = \frac{1}{n} \sum_k |f_k(\mathbf{x})|^2$, where $f_k(\mathbf{x})$ is the kth component of the discrete Fourier transform f $\mathbf{x}$.

The most obvious way to define superposition with circular vectors is as the angle of the rectangular sum of complex numbers. This requires storing both the angle and magnitude (or the real and the imaginary components) of the complex values during computation of the superposition. At the end of the computation the magnitudes are discarded. This corresponds to computing to the sum in the spatial domain and normalizing the frequency magnitudes of the result. It is possible to use weights in superpositions, by multiplying each vector by a weight before adding it into the superposition. Thus, we can define the superposition function sp() for k angles (i.e., complex values on the unit circle) with weights w_i as:

$$\mathrm{sp}(w_1, \ldots, w_k, \phi_1, \ldots, \phi_k) = \theta$$

$$\text{such that } \cos(\theta) = \sum_{i=1}^{k} w_i \cos\phi_i \text{ and } \sin(\theta) = \sum_{i=1}^{k} w_i \sin\phi_i.$$

This definition is easily extended to vectors of angles by performing the same operation with corresponding elements of the vectors. For notational convenience, I omit the weights when they are all the same. I use the symbol $\oplus$ to denote the superposition of two vectors: $\theta \oplus \phi = \mathrm{sp}(\theta, \phi)$.

This superposition operation differs from standard superposition in that it is not associative. In general,

$$((\theta \oplus \phi) \oplus \psi) \neq (\theta \oplus (\phi \oplus \psi)).$$

This is due to the normalization of magnitudes. However, this is not a serious problem, and the same is true for the standard system with normalization.

Table 4.1 summarizes the corresponding entities and operations for standard and circular systems. In order to predict how well circular vectors and the associated operations will perform as a basis for HRRs, we need to know the means and variances of the similarity and

Entity/Opertion	Circular system	Standard system
Random vector	Elements iid as $U(-\pi, \pi)$	Elements iid as $N(0, 1/n)$
Superposition	Angle of sum $\bar{\theta} \oplus \bar{\phi}$, sp()	Addition $\mathbf{x} + \mathbf{y}$
Weights in superposition	Weights in angle of sum	Scalar multiplication
Additive zero	No corresponding object	$\bar{0} = [0, 0, \ldots, 0]^T$
Association	Modulo-2π sum of angles $\bar{\theta} \odot \bar{\phi}$ (Elementwise complex multiplication)	Convolution $\mathbf{x} \circledast \mathbf{y}$
Associative identity	Vector of zero angles	Convolutive identity: $\bar{I} = [1, 0, \ldots, 0]^T$
Exact inverse	Negation (modulo-2π) $-\bar{\theta}$	Inverse $\mathbf{x}^{-1}$
Approximate inverse	Same as exact inverse	Involution $\mathbf{x}^*$
Similarity	Sum of cosine of differences $\bar{\theta} \cdot \bar{\phi}$	Dot-product $\mathbf{x} \cdot \mathbf{y}$
Normalization	Not needed – circular vectors are normalized	$\langle \mathbf{x} \rangle$

Table 4.1: Corresponding entities and operations in the circular and standard systems. "Iid" means "independently and identically distributed".

decoding operations. In Appendix H I derive means and variances for some combinations, and report experimental results. It turns out that the variances for decoding bindings of circular vectors are lower and more uniform than for normalized or unnormalized standard vectors, which means that the circular system should be less noisy. However, the means for dot-products of similar vectors (such as $\bar{\theta} \cdot (\bar{\theta} \oplus \bar{\phi})$) are also lower, which to some extent balances the advantage of lower variances. Tables 4.2 and 4.3 show the means and variances of dot-products of various expressions for circular and unnormalized standard vectors. In Appendix H, I also compare the circular system to the normalized standard system. Overall, it appears that the circular system is far superior to the unnormalized standard system, and slightly superior to the normalized standard system in terms of signal-to-noise ratios.

4.3 Comparison with the standard system

Overall, the circular system appears to be superior to the standard system. Advantages of the circular relative to the standard system are these:

- Variances of dot-products are lower, which means that there is less noise in decoding and in similarity comparisons.
- Variances of dot-products of convolution expressions are simpler and more uniform, which makes variances easier to calculate (compare Table H.1 in Appendix H with Table 3.1).
- The involution (which is the approximate inverse in the standard system) of a unitary circular vector is equal to its exact inverse.

	Expression	Circular system mean	Circular system variance	Standard system mean	Standard system variance
(1)	$\mathbf{a} \cdot \mathbf{a}$	1	0	1	$\frac{2}{n}$
(2)	$\mathbf{a} \cdot \mathbf{b}$	0	$\frac{1}{2n}$	0	$\frac{1}{n}$
(3)	$\mathbf{a} \cdot \mathbf{a} \odot \mathbf{b}$	0	$\frac{1}{2n}$	0	$\frac{2n+1}{n^2}$
(4)	$\mathbf{a} \cdot \mathbf{b} \odot \mathbf{c}$	0	$\frac{1}{2n}$	0	$\frac{1}{n}$
(5)	$\mathbf{a} \odot \mathbf{a} \cdot \mathbf{a} \odot \mathbf{a} = \mathbf{a} \odot \mathbf{a} \odot \mathbf{a}^* \cdot \mathbf{a}$	1	0	$2 + \frac{2}{n}$	$\frac{40n+112}{n^2}$
(6)	$\mathbf{a} \odot \mathbf{b} \cdot \mathbf{a} \odot \mathbf{b} = \mathbf{a} \odot \mathbf{b} \odot \mathbf{a}^* \cdot \mathbf{b}$	1	0	1	$\frac{6n+4}{n^2}$
(7)	$\mathbf{a} \odot \mathbf{b} \cdot \mathbf{a} \odot \mathbf{a} = \mathbf{a} \odot \mathbf{b} \odot \mathbf{a}^* \cdot \mathbf{a}$	0	$\frac{1}{2n}$	0	$\frac{6n+18}{n^2}$
(8)	$\mathbf{a} \odot \mathbf{b} \cdot \mathbf{a} \odot \mathbf{c} = \mathbf{a} \odot \mathbf{b} \odot \mathbf{a}^* \cdot \mathbf{c}$	0	$\frac{1}{2n}$	0	$\frac{2n+2}{n^2}$
(9)	$\mathbf{a} \odot \mathbf{b} \cdot \mathbf{c} \odot \mathbf{c} = \mathbf{a} \odot \mathbf{b} \odot \mathbf{c}^* \cdot \mathbf{c}$	0	$\frac{1}{2n}$	0	$\frac{2n+2}{n^2}$
(10)	$\mathbf{a} \odot \mathbf{b} \cdot \mathbf{c} \odot \mathbf{d} = \mathbf{a} \odot \mathbf{b} \odot \mathbf{c}^* \cdot \mathbf{d}$	0	$\frac{1}{2n}$	0	$\frac{1}{n}$

Table 4.2: Means and variances of dot-products for decoding and comparing bindings. The first two columns of numbers are for the circular system (reproduced from Table H.1), and the last two columns are for the corresponding operations (i.e., using $\circledast$ instead of $\odot$) in the unnormalized standard system (reproduced from Table 3.1). n is the dimensionality of the vectors. The dot-products are normally distributed in the limit as n goes to infinity.

Circular system			Standard (unnormalized) system		
Expression	mean	variance	Expression	mean	variance
$\mathbf{a} \cdot \mathbf{a}$	1	0	$\mathbf{a} \cdot \mathbf{a}$	1	$\frac{2}{n}$
$\mathbf{a} \cdot \mathbf{b}$	0	$\frac{1}{2n}$	$\mathbf{a} \cdot \mathbf{b}$	0	$\frac{1}{n}$
$\mathbf{a} \cdot (\mathbf{a} \oplus \mathbf{b})$	$\frac{2}{\pi} = 0.637\ldots$	$\frac{-8+\pi^2}{2\pi^2 n} = \frac{0.0947\ldots}{n}$	$\mathbf{a} \cdot \frac{1}{\sqrt{2}}(\mathbf{a} + \mathbf{b})$	$\frac{1}{\sqrt{2}}$	$\frac{3}{2n}$

Table 4.3: Means and variances for comparing similar, dissimilar, and half-similar vectors. $\mathbf{a}$ and $\mathbf{b}$ are random vectors. The expressions for the circular system are from Appendix H.

- It is very easy to quantize angles for lower precision representations, and create lookup tables for faster superposition, association and similarity computation.
- Encoding and decoding bindings can be done in $O(n)$ time, compared to $O(n \log n)$ time for the standard system.

Disadvantages of the circular relative to the standard system are these:

- The means of similarity comparisons are lower, but the lower variances compensate for this.
- Superposition is slower to compute, because the calculations are more complex. It still can be done in $O(n)$ time, but the constant is higher.
- The dot-product calculation is slower to compute because it involves n cosines. However, these could be computed by a fast table lookup, so this is not very important.

- The scalar values in the distributed representation are angles rather than magnitudes, which makes it more difficult to use with common neural networks.

The two system are equivalent with respect to scaling properties, because the variance of decoding and similarity operations is inversely proportional to n.

For the remainder of this thesis I use normalized standard vectors rather than circular vectors, for two reasons. The first is that I expect standard vectors to be more accessible to researchers familiar with popular connectionist representations. The second is that the computational advantage of using circular vectors is less than it might appear. Encoding and decoding bindings is faster: it takes $O(n)$ time, as opposed to $O(n \log n)$ for the standard representation. However, the computational cost of simulations tends to be dominated by the clean-up memory, and the circular vector similarity operation is more expensive to compute.[1] In any case, all the experiments in this thesis could be replicated with circular vectors, and neural networks can be easily adapted to work with complex numbers.

[1] It table lookup were used for the cosines, the amount of computation for clean-up would be the same in the two systems.

Chapter 5

Using convolution-based storage in systems which learn

Hinton [1986] and Miikkulainen and Dyer [1989] showed that useful distributed representations of objects could be learned using gradient descent in feedforward (backpropagation) network. Pollack [1990] showed that an auto-encoder network could even learn representations for compositional structures. This ability to learn representations has two important applications: developing new representations, and fine-tuning existing ones.

Given the difficulties encountered in learning representational schemes for compositional structures (such as Pollack's), and given that there are simple convolution-based methods for representing compositional structures, it is interesting to consider how convolution-based storage schemes might be incorporated into networks which could learn representations using gradient descent training. In this chapter I describe how the trajectory-association method for representing sequences (Section 3.4.1) can be incorporated into a recurrent network. I report some experiments in which these networks develop representations for objects and sequences, and compare their performance with that of more conventional recurrent networks.

5.1 Trajectory-association

A simple method for storing sequences using circular convolution is to associate elements of the sequence with points along a predetermined trajectory. This is akin to the memory aid called the method of loci which instructs us to remember a list of items by associating each term with a distinctive location along a familiar path. Elements of the sequence and loci (points) on the trajectory are all represented by n-dimensional vectors. For example, the sequence "abc" can be stored as:

$$\mathbf{s}_{abc} = \mathbf{p}_0 \circledast \mathbf{a} + \mathbf{p}_1 \circledast \mathbf{b} + \mathbf{p}_2 \circledast \mathbf{c},$$

where the $\mathbf{p}_i$ are vectors representing the loci, which should not be similar to each other, and $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$ are the vectors representing the items. This representation of a sequence can be decoded by convolving it with the approximate inverses of the $\mathbf{p}_i$ in the appropriate order.

An easy way to generate a suitable sequence of loci is to use convolutive powers of a random unitary vector $\mathbf{k}$, i.e., $\mathbf{p}_i = \mathbf{k}^i$. Convolutive powers are defined in the obvious

way: $\mathbf{k}^0$ is the identity vector and $\mathbf{k}^{i+1} = \mathbf{k}^i \circledast \mathbf{k}$. Unitary vectors (Section 3.6.3) have unit magnitude in each of their frequency components – thus the magnitude of each of their eigenvalues is one. They are the class of vectors for which the exact inverse is equal to the approximate inverse. Restricting $\mathbf{k}$ to be unitary ensures that $\mathbf{k}$ does not blow up or disappear when raised to high powers, because, for any unitary vector $\mathbf{k}$, and for all t, $||\mathbf{k}^t|| = 1$;

Using convolutive powers for the loci, the trajectory-association representation for the sequence "abc" is:

$$\mathbf{s}_{abc} = \mathbf{a} + \mathbf{b} \circledast \mathbf{k} + \mathbf{c} \circledast \mathbf{k}^2.$$

Raising a unitary vector to some power t corresponds to multiplying the phase angles of its frequency components by t. (Fractional convolutive powers are easily defined in this way.) The similarity of two unitary vectors depends on how well their corresponding phase angles match – the dot-product is proportional to the sum of the cosines of the differences of the angles:

$$\mathbf{x} \cdot \mathbf{y} = \frac{1}{n} \sum_j \cos(\theta_j(\mathbf{x}) - \theta_j(\mathbf{y})),$$

where $\theta_j(\mathbf{x})$ is the phase angle of the jth (frequency) component of the discrete Fourier transform of $\mathbf{x}$.[1] Since the phase angles are randomly distributed in $(-\pi, \pi]$, different integer powers of the same unitary vector have low expected similarity. Of course, for unitary $\mathbf{k}$ with phase angles that are rational fractions of π, there is some t such that $\mathbf{k}^t = \mathbf{k}$. However, this t is almost always very large. Figure 5.1 shows some plots of the similarity of $\mathbf{k}^t$ to $\mathbf{k}$, for unitary vectors of various dimensions. For higher dimensions, the similarity of $\mathbf{k}$ and $\mathbf{k}^t$ for $t >= 2$ has lower variance. Note that for t between 0 and 2, $\mathbf{k}$ and $\mathbf{k}^t$ become more similar as t becomes closer to 1. Also, it is easy to show that the similarity of two vectors is invariant under translation in power: $\mathbf{k} \cdot \mathbf{k}^t = \mathbf{k}^\alpha \cdot \mathbf{k}^{\alpha+t}$ for all α. This means that the similarity of $\mathbf{k}^6$ and $\mathbf{k}^8$ is the same as the similarity of $\mathbf{k}$ and $\mathbf{k}^3$, for example.

The waveforms in Figure 5.1 are actually the superpositions of $n/2-1$ equally weighted sinewaves, and a constant $2/n$.[2] Each sinewave is generated by one frequency component of $\mathbf{k}$, and its period is $2\pi/\alpha$, where α is the phase angle (in $(-\pi, \pi]$).

5.1.1 Decoding trajectory-associated sequences

Trajectory-associated sequences can be decoded by repeatedly convolving with the inverse of the vector that generated the encoding loci. As with all convolution-based associations, the results are noisy and must be cleaned up. This requires all possible items to be stored in the clean-up memory.

There are two exactly equivalent methods for decoding a trajectory-associated sequence, which are shown in Table 5.1. Since the approximate inverses of the loci are equal to their exact inverses (i.e., $(\mathbf{k}*)^i = \mathbf{k}^{-i}$), I use negative powers to indicate the inverses of the loci. In the direct access method, the inverses of each loci are used to probe the sequence representation:

$$\mathbf{r}_i = \mathbf{k}^{-i} \circledast \mathbf{s},$$

[1] This identity is easily proved by substituting $(\mathbf{z} - \mathbf{y})$ for $\mathbf{x}$ in the discrete version of Parseval's theorem, which states that $\sum_k x_k^2 = \frac{1}{n} \sum_k |f_k(\mathbf{x})|^2$, where $f_k(\mathbf{x})$ is the k component of the discrete Fourier transform f $\mathbf{x}$.

[2] The constant of $2/n$ comes from the two frequency components which have a phase angle of zero for real vectors.

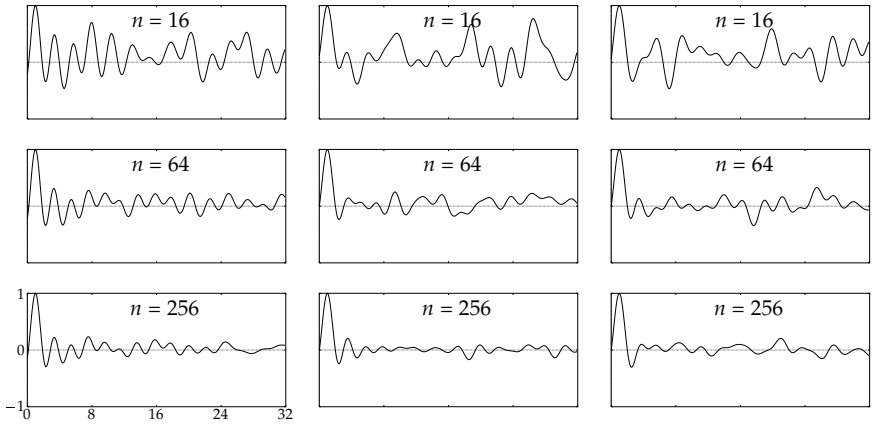


Figure 5.1: Example trajectory similarities. Each plot shows the similarity (vertical axis) of $\mathbf{k}$ to $\mathbf{k}^t$, for $t = 0$ to 32 (horizontal axis), for randomly selected unitary vectors $\mathbf{k}$ of dimension n. All plots are to the same scale. The first peak in each graph is for $t = 1$ (because $\mathbf{k}^1 = \mathbf{k}$). For large n, randomly chosen $\mathbf{k}$, and $t > 2$, the expected similarity of $\mathbf{k} \cdot \mathbf{k}^t$ is $2/n$, and the variance decreases with n.

Direct		Recursive		Expansion		Result
$\mathbf{k}^0 \circledast \mathbf{s}_{abc}$	=	$\mathbf{s}_{abc}$	=	$\mathbf{a} + \mathbf{b} \circledast \mathbf{k} + \mathbf{c} \circledast \mathbf{k}^2$	$\approx$	$\mathbf{a}$
$\mathbf{k}^{-1} \circledast \mathbf{s}_{abc}$	=	$\mathbf{k}^{-1} \circledast \mathbf{s}_{abc}$	=	$\mathbf{a} \circledast \mathbf{k}^{-1} + \mathbf{b} + \mathbf{c} \circledast \mathbf{k}$	$\approx$	$\mathbf{b}$
$\mathbf{k}^{-2} \circledast \mathbf{s}_{abc}$	=	$\mathbf{k}^{-1} \circledast (\mathbf{k}^{-1} \circledast \mathbf{s}_{abc})$	=	$\mathbf{a} \circledast \mathbf{k}^{-2} + \mathbf{b} \circledast \mathbf{k}^{-1} + \mathbf{c}$	$\approx$	$\mathbf{c}$

Table 5.1: The direct and recursive methods of decoding a trajectory-associated sequence.

where $\mathbf{r}_i$ is the decoded ith element of the sequence represented by $\mathbf{s}$. In the recursive access method, the sequence representation is repeatedly convolved with $\mathbf{k}^{-1}$, which steps through the elements of the sequence:

$$\mathbf{r}_0 = \mathbf{s} \quad \text{and} \quad \mathbf{r}_i = \mathbf{k}^{-1} \circledast \mathbf{r}_{i-1}$$

These two methods are equivalent because convolution is associative.

For example, to retrieve the third element of the sequence $\mathbf{s}_{abc}$ we convolve twice with $\mathbf{k}^{-1}$, and the result expands to $\mathbf{a} \circledast \mathbf{k}^{-2} + \mathbf{b} \circledast \mathbf{k}^{-1} + \mathbf{c}$. The first two terms are unlikely to be correlated with any items in the clean-up memory, so the most similar item will probably be $\mathbf{c}$.

5.1.2 Capacity of trajectory-association

In Appendix D, I calculate the capacity of (circular) convolution-based associative memory. I assume that the elements of all vectors (dimension n) are chosen randomly from a Gaussian distribution with mean zero and variance $1/n$ (giving an expected Euclidean length of 1.0).

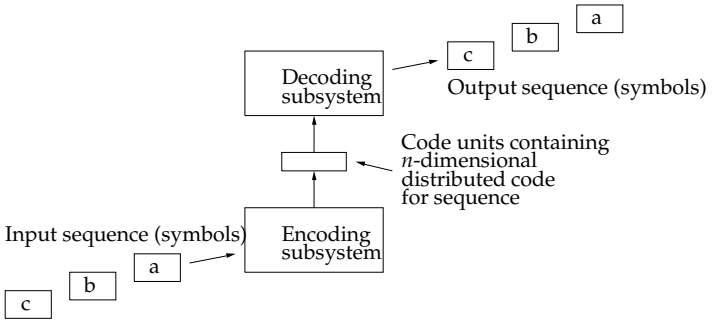


Figure 5.2: Subsystems to encode a sequence of symbols into a trajectory-associated code, and decode the code back into a sequence of symbols. The encoding and decoding subsystems must store internally the n-dimensional vectors which represent the symbols.

Even for short sequences, quite high vector dimensions are required to ensure a low probability of error in decoding. For example, with 512 element vectors and 1000 items in the clean-up memory, 4 pairs can be stored with about a 1% chance of an error in decoding. The scaling is nearly linear in n – with 1024 element vectors 8 pairs can be stored with about a 1% chance of error. This works out to an information capacity of about 0.16 bits per element. The elements are floating-point numbers, but high precision is not required.

These capacity calculations are roughly applicable to trajectory-association – the number of pairs corresponds to the length of sequences. They slightly underestimate its capacity because of the restriction that the encoding loci are unitary vectors, which results in lower decoding noise. Nonetheless this figure provides a useful benchmark against which to compare the capacity of the adaptive networks described in this chapter, which, on account of the training, should be able to store more.

5.2 Encoding and decoding systems for trajectory-associated sequences

Figure 5.2 shows the obvious way in which an encoding subsystem and a decoding subsystem can be composed to form a system which can store and recall sequences. In a system which uses trajectory-association to form distributed codes for sequences, there are at least two opportunities for learning representations. We can learn distributed representations for the individual items (à la Hinton [1986]), and we can fine-tune the distributed code for a sequence. The latter can be useful if errors occur during recall of the sequence – often we can adjust the code so that it is read out correctly.

If the error surfaces of the decoder are simple enough, the idea of adjusting the code for the sequence so that it is read out correctly can be extended to finding the code for a sequence from an arbitrary starting point. To do this, we instantiate the code units with some starting vector (e.g., a random vector), run the decoder to see what sequence it produces, compare that to the sequence we want to produce, backpropagate the errors, and adjust the code units to better produce the desired sequence. We repeat this until the code units contain the vector which produces the desired sequence.

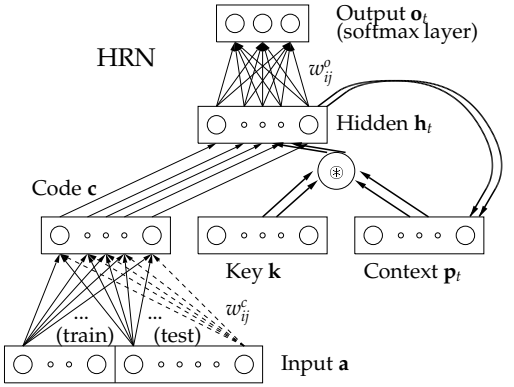


Figure 5.3: A Holographic recurrent network (HRN). The backwards curved arrows denote a copy of activations to the next timestep. The hidden, code, context, and key layers all have the same number of units. The left block of input units is used during training, the right block during testing.

In this chapter I only consider the decoding system, and investigate to what extent gradient descent methods can learn good representations for items and find codes for producing sequences of items. In some ways, this is the opposite of how recurrent networks are usually used to process sequences, in that the sequence is the output rather than the input. The reason I do this is that I wish to focus on representations for sequences. Decoding the sequence representation is a convenient way of ensuring that all the sequential information is present. Furthermore, at this preliminary stage I wish to avoid the complexity of constructing both encoding and decoding systems.

5.3 Trajectory-association decoding in a recurrent network

It is straightforward to build the trajectory-association decoding operations (Section 5.1.1) into a recurrent network. I call the resulting network a *Holographic recurrent network* (HRN).

5.3.1 Architecture

The HRN used in the experiments described here is shown in Figure 5.3. The clean-up memory is easily implemented with a "softmax" output layer (See Section 5.3.2). The output layer does not actually hold distributed representations for symbols, instead it uses a local representation. The softmax function ensures that the total activity in the output layer sums to 1, so that the output activities can be interpreted as probabilities. The output unit with maximum activation will be the one whose incoming weights best match the activations on the hidden units (the incoming weights on the outputs are the distributed representations of symbols). Using a local output representation and the softmax function allows us to take derivatives of the error function. If we wanted to have distributed representations for symbols on the output, we could add an extra output layer and the weights to map the local representation to the distributed one.

The key layer **k** has fixed activation and contains the generator for the inverse loci (**k** here corresponds to $\mathbf{k}^{-1}$ in Section 5.1, because **k** here is the decoding rather than the encoding key.).

The input representation is also local, only one input unit is active for each sequence. The outgoing weights from the single active input unit determine the activations at the code layer – these weights form a local-to-distributed map (as discussed in Section 2.2.2. The code layer injects activation into the hidden layer at the first step. After that the hidden layer activations are cycled through the context layer, so the input and code units no longer affect the network.

Each sequence is produced independently – no residual activations from the production of previous sequences remain in the network.

5.3.2 Unit functions

The HRN computes the following functions. The activations of the input, code and key layers remain constant during the production of a sequence. The parameter g is an adaptable input gain shared by all output units.

Code units:		$c_j = \sum_i a_i w^c_{ji}$	
Hidden units:	(first timestep)	$h_{1,j} = c_j$	
	(subsequent steps)	$h_{t,j} = \sum_i p_i k_{j-i}$	$(\mathbf{h} = \mathbf{p} \circledast \mathbf{k})$
Context units:		$p_{t+1,j} = h_{t,j}$	
Output units:	(total input)	$x_{t,j} = g \sum_i h_{t,i} w^o_{ji}$	
	(output)	$o_{t,j} = \frac{e^{x_{t,j}}}{\sum_i e^{x_{t,i}}}$	(softmax)

On all sequences, the net is cycled for as many timesteps as required to produce the target sequence. The outputs do not indicate when the net has reached the end of the sequence. However, other experiments have shown that it is a simple matter to add an extra output unit to indicate the end of the sequence.

The inputs and outputs to the network look like this:

		Output			
Sequence	Input	t = 0	t = 1	t = 2	t = 3
abc	1 0 0 00	1 0 0	0 1 0	0 0 1	
baac	0 1 0 00	0 1 0	1 0 0	1 0 0	0 0 1

5.3.3 Objective function

The objective function of the network is the asymmetric divergence between the activations of the output units ($o^s_{t,j}$) and the targets ($t^s_{t,j}$) summed over cases s and timesteps t, plus two weight penalty terms:

$$E = -\left(\sum_{stj} t^s_{t,j} \log \frac{o^s_{t,j}}{t^s_{t,j}}\right) + \frac{0.0001}{n} \sum_{jk} (w^c_{jk})^2 + \sum_j \left(1 - \sum_k (w^o_{jk})^2\right)^2$$

The first weight penalty term is a standard weight cost designed to penalize large weights (n is the number of hidden units). It is probably superfluous in the HRN, because the various operations from the w^c_{jk} to the total inputs of the output layer (x_j) are all linear, and scaling of the w^c_{jk} can be compensated for by the gain g. Later experiments without this weight cost term indicated that it has no effect on performance.

The second weight penalty term is designed to force the Euclidean length of the weight vector on each output unit to be one. This makes the output layer more like an ideal clean-up memory. The generalization performance with this special penalty term for the output weights is considerably better than that with a standard weight cost for the output weights (which is in turn better than that with no weight penalty). The reason for the worse performance with the standard weight cost is that the symbols which occur more frequently in the training set develop output weight vectors with higher magnitudes.

5.3.4 Derivatives

To do gradient descent learning, partial derivatives for all the adjustable parameters are required: $\frac{\partial E}{\partial w^o_{jk}}$, $\frac{\partial E}{\partial w^c_{jk}}$, and $\frac{\partial E}{\partial g}$. All of these can be easily calculated from the above equations using the chain rule and Rumelhart, Hinton, and William's [1986] unfolding-in-time technique. The only partial derivatives not encountered in common backpropagation networks are those for the activations of the context units:

$$\frac{\partial E}{\partial p_j} = \sum_k \frac{\partial E}{\partial h_j} y_{k-j} \qquad (\equiv \nabla_{\mathbf{p}} = \nabla_{\mathbf{h}} \circledast \mathbf{y}^*)$$

When there are a large number of hidden units it is more efficient to compute this derivative via FFTs as the convolution expression on the right ($\nabla_{\mathbf{p}}$ and $\nabla_{\mathbf{h}}$ are the vectors of partial derivatives of E with respect to **p** and **h**).

5.3.5 Training and testing productive capacity

To be useful, a trainable storage scheme for structures must be able to store a wide variety of novel structures without requiring adjustment of its basic parameters after initial training. In an HRN, the basic parameters are the weights on the output units. The novel structures are sequences the network was not exposed to in the initial training. I refer to the ability of the network to store novel sequences as its *productive capacity*.

I performed initial training on a small number of short sequences in order to find good values for the basic parameters: the output weights w^o_{jk} and the output gain g. The initial training also finds code unit instantiations which produce the training sequences, which are determined by the weights w^c_{ij} coming from the block of input units on the left in Figure 5.3. To estimate the productive capacity of HRNs I freeze these parameters and test whether the network is able to produce longer novel sequences. For each novel sequence, I perform an independent search for a vector of code unit activations which will cause the network to produce the sequence. These activations are determined by the weights w^c_{ij} coming from the block of input units on the right in Figure 5.3.

For the initial training I use randomly chosen sequences of length 4 over various numbers of symbols. The number of symbols corresponds to the number of output units in the network. The number of sequences in the training set is 4 times the number of symbols, which means that on average each symbol occurs 16 times in a training set. For example, there are 12 sequences in the training set over three symbols. I initialize all weights to small random values, and the output gain to 1. I use conjugate gradient minimization and stop when all sequences were correctly produced (or when 400 steps have been taken). I judge a sequence to be correctly produced only if all its symbols are correct. I judge a symbol to

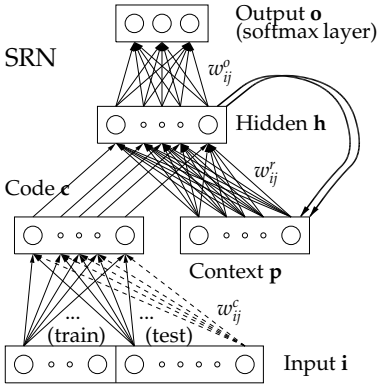


Figure 5.4: A simple recurrent network (SRN). The backwards curved arrows denote a copy of activations to the next timestep. The hidden, code, and context layers all have the same number of units. The left block of input units is used during training, the right block during testing.

be correct when the activation of the correct output unit exceeds 0.5 and exceeds twice any other output unit activation.

For estimating the productive capacity I use novel sequences of lengths 3 to 16 (32 sequences of each length). I use a separate conjugate gradient search to find a code for each sequence (starting from small random values), and stop when the sequence has been correctly produced, or when 100 steps have been taken.

5.4 Simple recurrent networks

I compare HRNs to other more conventional recurrent networks, both to get some idea of their relative capacities and because the comparison gives some insight into various reported difficulties with recurrent networks. Elman [1991] and Servan-Schreiber, Cleeremans and McClelland [1991] use Simple Recurrent Networks (SRNs) to process sequential input and induce small finite-state grammars. However, their training times are extremely long, even for very simple grammars. Maskara and Noetzel [1992] report similar results and make the plausible suggestion that the long training times are due to the difficulty of finding a recurrent operation that preserves information in the hidden units over many timesteps. Simard and Le Cun [1992] report being unable to train a type of recurrent network to produce more than one trajectory through continuous space, a task which requires careful control over information in the hidden units.

Given these reported difficulties, I expected that SRNs would perform poorly at the sequence production task, since it requires preserving and transforming information in the hidden units over many timesteps. To test this, I built a version of the SRN for the sequence production task. The architecture of this SRN differs from the HRN in that the hidden layer receives activation through a weight matrix from the context layer, as with the SRNs. The network, which I refer to as a SRN, is shown in Figure 5.4.

The only difference between the SRN and the HRN is in the computation of the activations of the hidden units:

$$h_j = \tanh(c_j + \textstyle\sum_k w^r_{jk} p_k + b_j),$$

where b_j is a bias on each hidden unit. As with the HRN, the activations on the code units do not change during the production of a sequence. However, in contrast with the HRN, the activations of the code units flow to the hidden units at every timestep.

The objective function for the SRN is the same as for the HRN, except for the inclusion of an extra penalty term for large recurrent weights (i.e., w^r_{ij}, the weights from the context to the hidden layer):

$$E = -\left(\sum_{stj} t^s_{t,j} \log \frac{o^s_{t,j}}{t^s_{t,j}}\right) + \frac{0.0001}{n}\left(\sum_{jk}(w^r_{jk})^2 + \sum_{jk}(w^c_{jk})^2\right) + \sum_j \left(1 - \sum_k (w^o_{jk})^2\right)^2$$

The SRNs described here differ in two important ways from other SRNs described in the literature (besides being used to produce rather than process sequences): the tanh function (symmetric sigmoid) is used on the hidden units, rather than the non-symmetric logistic function, and full unfolding-in-time is used to calculate correct derivatives, rather than Elman's truncated derivative scheme. Experiments showed that both changes significantly improved training times and productive capacity.

In the simulations, I use almost exactly the same training and testing regime as for HRNs. The only difference is that I experimented with larger training sets.

5.5 Training and productive capacity results

5.5.1 HRN results

HRNs prove to be easy to train, and to have good productive capacity. They also appear to have good scaling properties – as the number of hidden units increases they can store longer sequences or sequences over more symbols. Figure 5.5 shows the performance of HRNs with eight output units and varying numbers of hidden units. The points shown are averages of data from five runs, each starting with a randomization of all parameters. As the number of hidden units increases from 4 to 64 the productive capacity increases steadily. For a HRN with a fixed number of hidden units, the length of sequences that can be stored reliably drops as the number of symbols increases, as shown in Figure 5.6. This is expected because when there are more symbols, more information is required to specify them. For example, the same amount of information is required to specify a sequence of length eight over three symbols as a sequence of length four over nine symbols.

To make a rough guess at the information efficiency of HRN sequence codes we can calculation how much information is in the maximum-length sequence that can be reliably encoded. For example, Figure 5.5 shows that a HRN with 16 hidden units can reliably store sequences of length 5 over 8 symbols. Such a sequence contains 15 bits of information (assuming all symbols are equally likely), which gives an information efficiency of nearly 1 bit per code element (15/16). This figure appears stable for different numbers of hidden units and for different numbers of symbols. This compares well with the 0.16 bit per element achieved by random vector circular convolution (Section 5.1.2). The increase in capacity can be attributed largely to the adaptive method of forming codes.

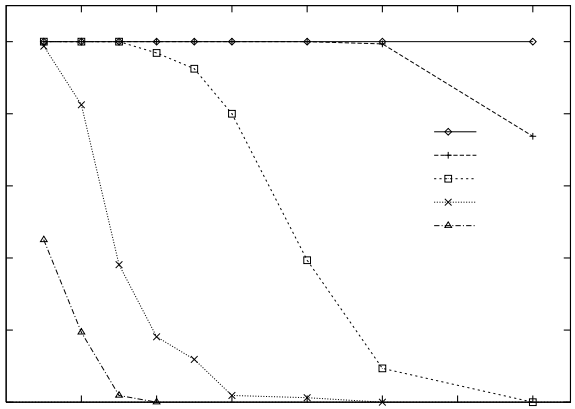


Figure 5.5: Percentage of novel sequences that can be produced, versus length of those sequences for HRNs with varying numbers of hidden unit. The sequences are over 8 symbols. All points are averages of data from 5 different training runs.

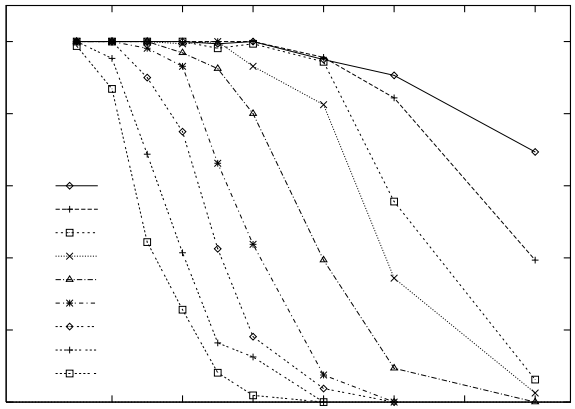


Figure 5.6: Percentage of novel sequences that can be produced by HRNs with 16 hidden units, versus length of those sequences, for varying numbers of symbols (i.e., varying numbers of output units). All points are averages of data from 5 different training runs.

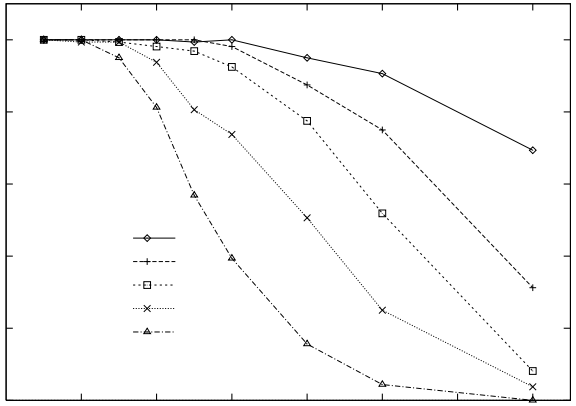


Figure 5.7: Percentage of novel sequences that can be produced, versus length of those sequences, for various types of networks (all with 16 hidden units, and 3 symbols). "SRN+" is the SRN with 8 times the as much training data and "SRN++" is the SRN with 32 times the as much. "SRNZ" is the SRN with frozen random recurrent weights. All points are averages of data from 5 different training runs.

5.5.2 SRN results

SRNs turned out to be better at this task than expected. Figure 5.7 shows how the performance varies with sequence length for various networks with 16 hidden units. SRNs required much more training data, which is not surprising since they have more parameters in the form of weights from the context to the hidden layer. The productive capacity for a SRN trained on 12 sequences of length 4 (the same amount of data as a HRN) was very poor – compare the bottom and top lines in the graph. However, with more training data consisting of longer sequences, the productive capacity was better, though never as high as that of HRNs. In the Figure, "SRN+" is the productive capacity of an SRN trained on 48 sequences of length 8 (8 times as much training data) and "SRN++" is one trained on 192 sequences of length 8 (32 times as much training data). The length of training sequences appeared to have a significant effect on productive capacity. Experiments not shown here indicated that the increase in productive capacity obtained by doubling the length of training sequences (4 to 8) was greater that obtained by doubling the number of training sequences.

Interestingly, freezing the recurrent weights at random values results in a significant increase in productive capacity for a SRN trained on the small training set ("SRNZ" in the Figure). Random values for all the weights coming into the hidden units are chosen from a Gaussian distribution with mean zero and the same variance as those from a sample trained SRN (0.2 for a net with 16 hidden units). These weights are frozen at the beginning of training and never changed. The productive capacity achieved with these weights tends to suggest that the recurrent weights are not doing much more than a random mapping.

The performance would probably improve further if the weight matrix were chosen so that all its eigenvalues were of equal magnitude (like the recurrent operation in HRNs).

In general, modifications to SRNs which make them more like HRNs seem to improve performance. As mentioned before, early experiments with the standard non-symmetric sigmoid operation on the hidden layer had very poor performance. The symmetric sigmoid (tanh), which is more like the linear transfer function in the HRN, gives much better performance. Freezing the hidden weights at random values also improves performance (for nets with a small training set). Other experiments showed that removing the bias on the hidden units usually improves performance. Changing the hidden unit transfer function of the SRN from tanh to linear improves performance on shorter novel sequences, but not on longer novel sequences. This is probably because the linear transfer function allows eigenvectors with large eigenvalues to grow exponentially, whereas the tanh function limits their growth. The only change that makes the SRN more like the HRN, and that consistently harms performance, is making the code units inject activation into the hidden units at the first timestep only.

5.5.3 General training and testing issues

The training times for both the HRNs and the SRNs are very short. Both require around 30 passes through the training data to train the output and recurrent weights. Finding a code for test sequences of length 8 takes the HRN an average of 14 passes, and the SRN an average of 57 passes (44 with frozen recurrent weights). The SRN trained on more data takes much longer for the initial training (average 281 passes) but the code search is shorter (average 31 passes).

Local minima in the space of codes for trained networks are not a significant problem for either HRNs or SRNs. Experiments with restarts for failed searches for a sequence code (during testing) show that allowing one restart improves the productive capacity figures by only 0.3% for HRNs and 1.3% for SRNs.

5.6 Trajectories in continuous space

Simard and Le Cun [1992] devise a "Reverse Time-Delay Neural Network" for producing trajectories through a continuous space. They train the network to produce pen trajectories for writing the characters "A" to "Z". One reported motivation for the complex architecture of their network is the difficulty they had with training a simpler recurrent network to produce more than one trajectory. HRNs can be adapted to produce trajectories, and have a much simpler architecture than Reverse Time-Delay Neural Networks. Only two modifications need be made: (a) change the function on the output units to sigmoid and add biases, and (b) use a fractional power of a random vector for the key vector. The hidden unit activations are still $\mathbf{k} \circledast \mathbf{c}$, where $\mathbf{c}$ is the code vector. The fractional power key vector $\mathbf{k}$ is derived by taking a random unitary vector $\mathbf{x}$ and multiplying the phase angle of each frequency component by some fraction α, i.e., $\mathbf{k} = \mathbf{x}^{\alpha}$. The result is that $\mathbf{k}^i$ is similar to $\mathbf{k}^j$ when the difference between i and j is less than $1/\alpha$, and the similarity is greater when i and j are closer. This is good for generating smooth trajectories because the decoding key at one step is similar to the decoding key at the next step. This means that the contents of the hidden layer will be similar at successive timesteps. The rate at which the output trajectory is traversed can be adjusted by changing α – smaller values result in a slower rate of change.

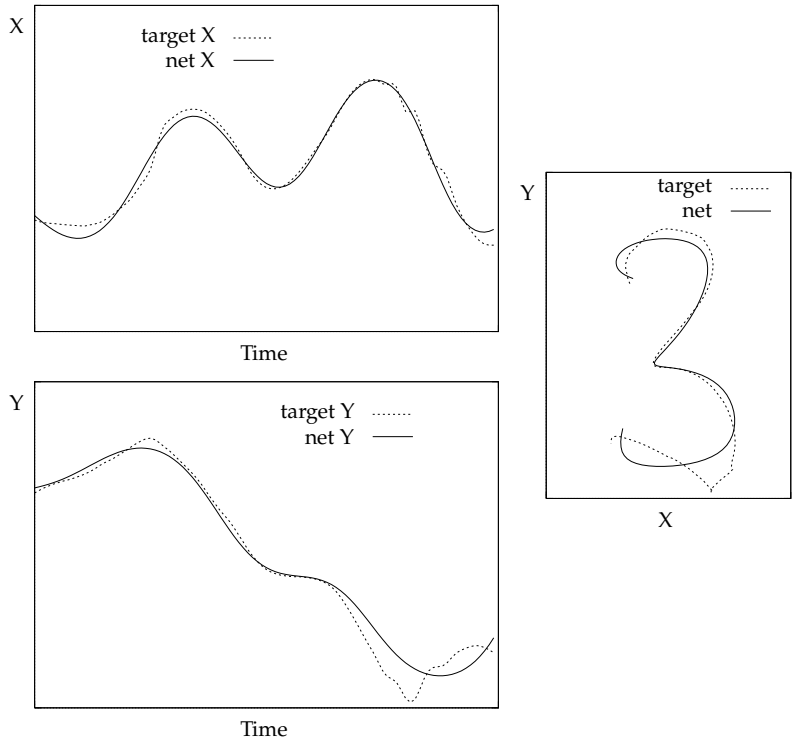


Figure 5.8: Targets and outputs of a HRN trained to produce trajectories through continuous space. The trajectory was not one that the network was trained on.

I tested a trajectory producing HRN on a task similar to Simard and Le Cun's: producing pen trajectories for the digits zero to nine. The HRN has 16 hidden units and a key vector with $\alpha = 0.06$. The trajectories have 100 steps in three dimensions (X, Y, and pen up/down). As with other HRNs, there are two sets of parameters: the input-to-code weights, and the output weights. I trained the network on 20 instances of handwritten digits and then froze the output weights. With the output weights frozen, the network could easily find codes that produce trajectories for novel digit instances. Figure 5.8 shows the X and Y target trajectories and the output of the network for one instance. The HRN does not reproduce trajectories exactly – the code has a very limited number of parameters and can be expected to impose some sort of regularization on the trajectory.

The code that produces a trajectory is a 16-element vector. One would expect the codes which produce similar trajectories to be similar. This does seem to be the case: 40 codes for producing the trajectories for 40 digits clustered into the digit classes with only one exception. The code vector is potentially useful wherever a compact representation for trajectories is needed, e.g., in classification.

It is possible to vary the rate of trajectory traversal by adding a unit to the network which exponentiates the key vector. Let ζ be the output of this unit. The hidden layer activations will be $\mathbf{k}^{\zeta} \circledast \mathbf{c}$. The network can be set up so that the value of ζ changes during the production of a trajectory. The base-line value for ζ should be one. To progress more rapidly through the trajectory ζ should be greater than one, and to progress more slowly, ζ should be less than one. It is not difficult to calculate partial derivatives for ζ, and preliminary experiments showed that a HRN could learn to control ζ dynamically so as to align the network output with a target sequence.

5.7 Hierarchical HRNs

The main attraction of circular convolution as an associative memory operator is its potential for representing hierarchical structure. In order to investigate whether gradient descent training could be used to find hierarchical codes for sequences I built a two-level HRN. It consists of two HRNs with the outputs of one (the main HRN) supplying the code for the other (the sub-HRN). Figure 5.9 shows the overall architecture of this system. The subsequence decoder (at the top of the figure) produces sequences of symbols from the codes it receives from the main sequence generator. When the subsequence decoder has finished a subsequence it signals the main sequence decoder to produce the next subsequence code. Each decoder is similar to one of the HRNs already described. The clean-up memory for the subsequence decoder, i.e., its output units, contains symbol codes. The clean-up memory for main sequence decoder contains subsequence codes. The clean-up memory uses a local representation, so the output of the main sequence decoder must be transformed back to a distributed representation (the subsequence codes) by a local-to-distributed mapping.

For example, suppose the encoding key for subsequence generator is $\mathbf{k}$, and the key for the main sequence generator is $\mathbf{j}$. Suppose that "abc", "de", and "fgh" are chunks the main sequence generator knows about. Then the sequence "abcdefgh" can be stored as

$$(\mathbf{a} + \mathbf{b} \circledast \mathbf{k} + \mathbf{c} \circledast \mathbf{k}^2) + (\mathbf{d} + \mathbf{e} \circledast \mathbf{k}) \circledast \mathbf{j} + (\mathbf{f} + \mathbf{g} \circledast \mathbf{k} + \mathbf{h} \circledast \mathbf{k}^2) \circledast \mathbf{j}^2.$$

This two-level HRN was able to produce sequences given appropriate codes. However the network was not able to find main sequence codes by using gradient descent with backpropagated errors from target symbols – there were too many local minima. This was the case even when the components of the network were trained separately and frozen. The local minima are most likely due to the presence of an intermediate clean-up memory.

5.8 Discussion

5.8.1 Application to processing sequential input

An issue in processing sequential data with neural networks is how to present the inputs to the network. One approach has been to use a fixed window on the sequence, e.g., as in Sejnowski and Rosenberg's [1986] NETtalk. A disadvantage of this is that any fixed size of window may not be large enough in some situations. Instead of a fixed window, Elman [1991] and Cleeremans *et al* [1991] used a recurrent net to retain information about previous inputs. A disadvantage of this is the difficulty that recurrent nets have in retaining information over many timesteps. HRNs offer another approach: use the codes that produce a sequence as input rather than the raw sequence. This would allow a fixed-size

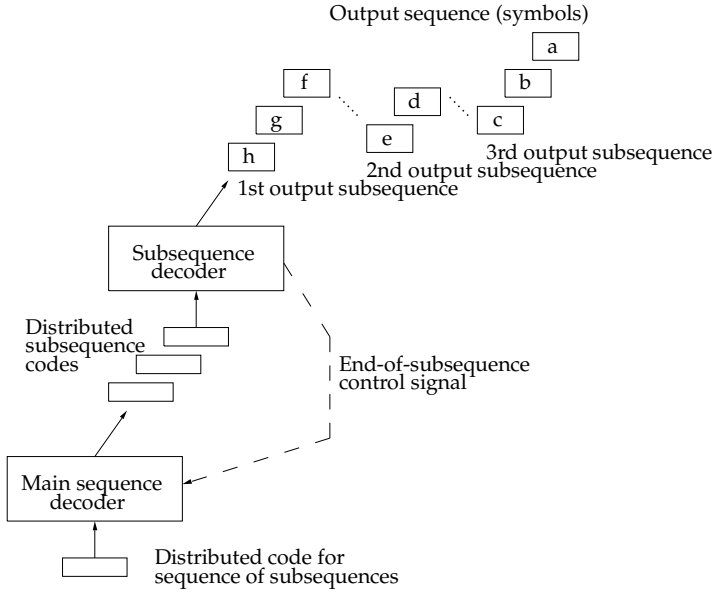


Figure 5.9: A two-level system for decoding nested sequences. The main system outputs a series of codes for subsequences, which are input to the subsequence decoder. When the subsequence decoder has completed a subsequence it signals the main sequence decoder to produce the next subsequence.

network to take sequences of variable length as inputs (as long as they were finite), without having to use multiple input blocks or windows. This would not be immediately applicable to networks which process sequences without any beginning or end, but some modification could be made along the lines of using a code which produces the recent relevant inputs.

5.8.2 Application to classification of sequences

The codes that produce sequences have similarity structure, e.g., the code for producing "abc" is more similar to the code for "acc" than the code for "cab". This also applies to the codes for producing trajectories through continuous space.

5.8.3 Improving the capacity of convolution-based memory

Adapting the representations of objects and traces gave a five-fold increase in the storage capacity of the trajectory-association method. Experiments with networks having many more output units showed that the information capacity was not affected by the number of different symbols used in the sequences.

The improvement in capacity is encouraging because it suggests that the capacity of other types of convolution-based memory could be improved by adapting representations, and also that traces that fail to decode correctly could be corrected by a simple gradient descent procedure.

5.9 Conclusion

It is straightforward to build the decoding operations for trajectory-associated sequences into a recurrent network. The resulting HRN can be trained using standard neural network methods. During training, the network simultaneously discovers memory traces for sequences and good representations for the objects. My experiments with HRNs demonstrate four points:

1. Gradient descent techniques are fast and reliable for learning representations for non-nested sequences from random starting points.
2. Gradient descent techniques are impractical for learning representations for nested sequences from random starting points.
3. Adapting memory traces and object representations increases the storage capacity of convolution-based memories.
4. For the task of producing sequences and trajectories, HRNs are as good as, if not better than, more conventional recurrent networks. They have fewer parameters, and consequently require less training data.

The convolution operation appears to be ideal for the recurrent operation in situations where information must be preserved but transformed in the hidden layer, because all the eigenvalues of the recurrent operation have a magnitude of one.

Chapter 6

Estimating analogical similarity

One of the advantages of reduced representations is the potential for using them in their reduced form. Such computations could be very fast, since they would not require time-consuming reconstruction of the full representation. However, the power and usefulness of this type of computation is limited by what information is made explicit in the reduced representation.

Similarity judgement is one potentially useful computation which could be performed with reduced representations. The vector dot-product[1] is an efficiently computable measure of similarity. The usefulness of this computation depends on the usefulness of the similarity structure this measure induces over structured objects. This in turn depends upon which aspects of the objects and structure are made explicit in the reduced representations under consideration.

In this chapter I investigate the similarity structure induced on structured objects by dot-product comparisons of Holographic Reduced Representations. It turns out that HRR dot-product similarity captures some aspects of analogical similarity as embodied in two models of analogical reasoning: ARCS (Analog Retrieval by Constraint Satisfaction) [Thagard et al. 1990] and SME (Structure Mapping Engine) [Falkenhainer, Forbus and Gentner 1989]. I give a set of examples which illustrate which aspects of similarity HRR dot-products are sensitive to. I also describe how some failings of HRR dot-product similarity as a measure of analogical similarity can be corrected by the inclusion of further bindings in the HRRs. ARCS is also a model of analog retrieval (from long-term memory), and MAC/FAC is a model of analog retrieval which uses SME for evaluation and reasoning. ARCS and MAC/FAC both require a fast method for estimating structural similarity in order to avoid expensive structural isomorphism tests on non-contenders. The methods they use for this are rudimentary and insensitive to structure – dot-product comparison of HRRs could provide a more discriminating but still efficient alternative.[2]

6.1 Models of Analogy Retrieval

Gentner and Markman [1993] suggested that the ability to deal with analogy will be a "watershed or Waterloo" for connectionist models. They identified "structural alignment", which is the matching of corresponding components of two structures in a manner which

[1] For binary representations the vector dot-product is the same as overlap.

[2] Whether or not this would make ARCS or MAC/FAC better theories is another issue, but the existence of a fast structure-sensitive estimate of similarity could open up new lines of investigation.

preserves the relationships, as the central aspect of analogy making, and noted the apparent ease with which people can do this in a wide variety of tasks. Gentner and Markman wondered whether surface forms of connectionist representation of structured objects would be similar to the degree that the underlying structures were similar, but expressed pessimism about the prospects for the development of such a model.

In fact, Holographic Reduced Representations provide such a representation. Among other things, HRRs can be used to obtain fast estimates of structural similarity. As we have seen, a HRR is a high-dimensional vector. The vector dot-product of two HRRs is an efficiently computable estimate of the overall similarity between the two structures represented. This estimate is sensitive to surface similarity and some, but not all, aspects of structural similarity,[3] even though correspondences between structure elements are not explicitly computed. Comparisons of HRRs can be made more sensitive to structural similarity by including additional bindings in the HRRs, which are derived using a technique I call "contextualization".

Gentner and Markman [1993] and Gentner, Markman and Wisniewski [1993] remark that people appear to perform structural alignment in a wide variety of tasks, including perception, problem solving, and memory recall. One task they and others have investigated is analog recall. A subject reads a number of stories and later reads a probe story. The task is to recall stories that are similar to the probe story (and sometimes evaluate the degree of similarity and perform analogical reasoning). This task is often thought to involve two stages – first is recall from long-term memory of stories that are potentially relevant and analogical to the probe story, and second is evaluation and use of the analogy, which involves construction of mappings between the recalled stories and the probe story. Two computer models of this process, Thagard *et al*'s [1990] ARCS and Gentner and Forbus's [1991] MAC/FAC (Many Are Called, Few Are Chosen), also have two stages. The first stage selects a few likely analogs from a large number of potential analogs. The second stage searches for an optimal (or at least good) mapping between each selected story and the probe story and outputs those with the best mappings. MAC/FAC uses Falkenhainer, Forbus and Gentner's [1989] "Structure Mapping Engine" (SME) as its second stage. The two-stage architecture is illustrated in Figure 6.1. Two stages are necessary because it is too computationally expensive to search for an optimal mapping between the probe and all stories in memory. An important requirement for a first stage is that its performance scale well with both the size and number of episodes in long-term memory. This requirement prevented the authors of MAC/FAC and ARCS from having the first stage consider structural features, because they knew of no efficient way to do this.

The evidence that people take structural correspondences into account when evaluating and using analogies is longstanding and solid (see Gentner, Rattermann and Forbus [1993] for a review). However, it is less certain whether structural similarity influences access to long-term memory (i.e., the first-stage reminding process). Gentner and Forbus [1991] and Gentner, Rattermann and Forbus [1993], found little or no effect of analogical similarity on reminding. Others, e.g., Wharton *et al* [to appear], Seifert and Hammond [1989] and Ross [1989], have found some effect. Wharton *et al* [to appear] suggest that the experimental design in earlier studies made it difficult to separate the effects of surface and structural similarity. In any case, surface features appear to influence the likelihood of a reminding far more than do structural features. Ross's [1989] study suggested that the effect of structural

[3] "Surface features" of stories are the features of the entities and relations involved, and "structural features" are the relationships among the relations and entities.

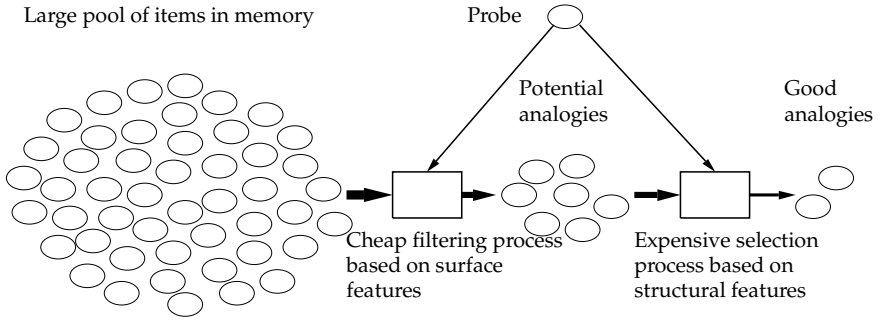


Figure 6.1: Two-stage models of analogy retrieval. The first stage selects potential analogies to the probe from the large pool of items in long-term memory, using fast but superficial comparisons. The bottleneck depicted here is not necessary – the computation required by the first stage could happen in parallel throughout long-term memory (e.g., in a content-addressable memory). The second stage performs a more expensive structural comparison between the probe and each potential analogy passed by the first stage.

similarity on reminding is greater in the presence of surface similarity.

The first goal of this chapter is to investigate how the dot-product of HRRs performs as an estimate of analogical similarity and to spell out the strengths and weaknesses of this estimate. To this end I show how dot-products of HRRs respond to various surface and structural aspects of similarity and compare the similarity judgements to those of ARCS and SME, two widely cited models of analog evaluation and analogical reasoning. ARCS (Analog Retrieval by Constraint Satisfaction) [Thagard et al. 1990] builds a local connectionist constraint satisfaction network (in its second stage) whose stable states are good interpretations of analogy between two structures. An interpretation of an analogy is a mapping of the entities in one structure to those in the other. The network is allowed to settle to a stable state and an overall similarity score is derived from how well the various constraints are satisfied. SME [Falkenhainer, Forbus and Gentner 1989] incorporates an explicit search for optimal analogical mappings. Both ARCS and SME provide an explicit interpretation of an analogy as well as a measure of goodness, and both can use the interpretation to perform analogical reasoning.

The second goal of this chapter is to suggest that dot-products of HRRs could be an efficient mechanism for the first stage of a model of analog retrieval. This would be more a powerful mechanism than the current first stages of both ARCS and MAC/FAC because it would be sensitive to structural as well as surface similarity.

However, a second stage would still be required for accurate evaluations of structural similarity, because the HRR dot-product is not an infallible measure of structural similarity. Furthermore, analogical reasoning tasks require an interpretation of an analogy, and this is not provided by HRR dot-products.

I use two examples to illustrate how analogical similarity can be estimated by dot-products of HRRs. The first is a set of shape configurations which can be represented by HRRs containing one relation. I include this set because it is a good illustration of some of the surface and simple structural features that people take into account when judging similarity. The similarity rankings given by HRR dot-products are the same as

those given by people. The second example consists a set of one-sentence "stories", which are represented by two or three relations in a nested HRR. These examples illustrate the extent to which HRRs are sensitive to the more complex aspects of structural similarity, such as patterns of variable instantiation and structural arrangement of predicates. The simple HRRs as described in Chapter 3 are shown to be insensitive to patterns of variable instantiation, but contextualizing HRRs corrects this deficit.

6.2 An experiment with shape configurations

Gentner, Markman, and Wisniewski [1993] report a simple experiment which demonstrates the importance of relational commonalities to human judgements of similarity. They asked subjects to judge the relative similarity of the pairs of configurations of geometric shapes shown in Table 6.1.

Subjects were shown configuration X and then asked to say which of A or B was more similar to it. A and B were presented in a random order to each subject. The elements of each comparison are systematically varied in order to find out what aspects of the relations are important to similarity judgements. The item most frequently selected as more similar to X is shown in column A in all cases, and the number of times it was selected is shown beside.

Gentner, Markman, and Wisniewski made the following comments about the stimuli:

> Item 1 suggests that having similar objects makes a pair of configurations similar, because only the similarity of the objects distinguishes between the two comparison figures. Items 2a and 2b suggest that having similar relations makes a pair similar regardless of whether the objects themselves are also similar. Item 3 suggests that having similar objects playing similar relational roles (e.g. both triangles on top) makes a pair similar. Items 4a and 4b suggest that any time a similar object plays a similar role it increases similarity, even if all of the objects taking part in the relation are not similar. Item 5 extends this finding by demonstrating that multiple relations can be used in the same comparison, as long as they give rise to the same object correspondences. In the most frequently selected alternative for this stimulus, the triangle is the top item and the smaller item, just as it is in the standard. Finally, Item 6 suggests that multiple relations can be matched, even when the objects making up the relations are dissimilar.
>
> This simple study makes four central points. First, both relational similarities and object similarities contribute to similarity. Second, a relational match can be made whether or not the objects that make up the relation also match. Third, similarity is sensitive to the *bindings* between relations and objects that make up the relation. Finally, many different relations can contribute to similarity as long as these relational matches give rise to the same object correspondences.

6.2.1 HRRs and dot-product similarity estimates

I constructed simple Holographic Reduced Representations for each of the configurations, and then obtained estimates of similarity by computing the dot-product of the HRRs. In all cases the HRR similarity estimates give the same judgements of relative similarity as the human subjects. The HRR dot-products (with 512-dimensional vectors) are shown in parentheses in Table 6.1 (e.g., 0.643 is the dot-product of the HRRs for A and X in the first row).

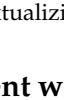
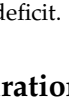
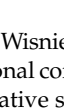
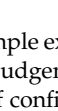
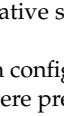
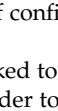
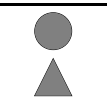
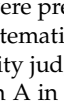
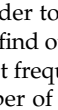
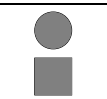
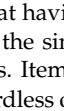
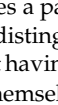
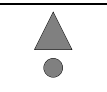
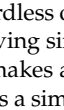
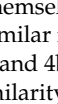
	X	A		B	
1			10 (0.545) [0.506 ± 0.032]		0 (0.347) [0.335 ± 0.034]
2a			8 (0.850) [0.837 ± 0.017]		2 (0.529) [0.506 ± 0.032]
2b			10 (0.656) [0.666 ± 0.023]		0 (0.367) [0.335 ± 0.036]
3			10 (1.000) [1.000 ± 0.000]		0 (0.850) [0.837 ± 0.017]
4a			9 (0.817) [0.831 ± 0.014]		1 (0.753) [0.751 ± 0.020]
4b			9 (0.858) [0.834 ± 0.017]		1 (0.745) [0.753 ± 0.019]
5			9 (1.000) [1.000 ± 0.000]		1 (0.837) [0.836 ± 0.014]
6			10 (0.569) [0.578 ± 0.028]		0 (0.369) [0.415 ± 0.036]

Table 6.1: Shape-configuration-matching materials and results from Markman *et al* [1993]. The number of respondents who judged A or B to be more similar to X is shown with each row. The HRR dot-product similarity estimate (from one run) of A and B with X is shown in parentheses. The vector dimension was 512. The means and standard errors over 100 runs are shown in square brackets.

Some of the HRRs are shown in Table 6.2. Two predicates are used. One is "are horizontal", which has two roles: "left" and "right". The other predicate is "are vertical", which has two roles: "above" and "below". These predicate names and roles are represented by the vectors **horizontal**, **left**, **right**, **vertical**, **above**, and **below**. All of these are "random base vectors" (vectors with elements chosen independently from $N(0, 1/n)$). The vectors representing the shapes are derived from the random base vectors **large**, **small**, **triangle**, **circle**, **square**, and **star** as follows ($\langle\cdot\rangle$ is the normalization operation):

	⟨⟨**lgCircle** + **lgSquare**⟩ + ⟨**vertical** + **above** ⊛ **lgCircle** + **below** ⊛ **lgSquare**⟩⟩
	⟨⟨**lgCircle** + **lgTriangle**⟩ + ⟨**horizontal** + **left** ⊛ **lgCircle** + **right** ⊛ **lgTriangle**⟩⟩
	⟨⟨**smStar** + **lgSquare**⟩ + ⟨**vertical** + **above** ⊛ **smStar** + **below** ⊛ **lgSquare**⟩⟩

Table 6.2: HRRs for some of the shape configurations. All variables are 512-dimensional vectors.

smTriangle = ⟨**small** + **triangle**⟩	**lgTriangle** = ⟨**large** + **triangle**⟩
smCircle = ⟨**small** + **circle**⟩	**lgCircle** = ⟨**large** + **circle**⟩
smSquare = ⟨**small** + **square**⟩	**lgSquare** = ⟨**large** + **square**⟩
smStar = ⟨**small** + **star**⟩	**lgStar** = ⟨**large** + **star**⟩

In these representations I treat size and shape as features and superimpose them to represent an object of a particular shape and size. This can create problems if the representations for objects are superimposed, because ownership of features becomes ambiguous (as discussed in Section 3.5.1). For example, the superposition **lgCircle** + **smSquare** is the same as **smCircle** + **lgSquare**. However, this problem does not arise in these examples. The representation for "large circle above small square" is not the same as the representation for "small circle above large square", because the size features ("small" and "large") are bound to the roles "above" and "below".

I did not represent Items 5 and 6 with multiple predicates, contrary to Markman *et al*'s interpretation of their significance. In fact, if the smallness and largeness of the shapes in Items 5 and 6 were represented with a predicate "is larger than" rather than features, and simple HRRs constructed, the resulting dot-product ranking for Item 6 would not agree with the human judgements, unless contextualization was used. I discuss the relationship between features and predicates in more depth in Section 6.8.3.

6.3 Analogies between multiple relations

To investigate how well the dot-product of HRRs estimates the similarity of complex descriptions I used a set of stories each involving two or three nested relations. Because these stories are so simple, I prefer to refer to them as "episodes." The probe and the "memorized" episodes are shown in Table 6.3. The memorized episodes are similar in different ways to the probe episode. These episodes are adapted from an example in Thagard *et al* [1990]. I added higher-order relational structure (the "cause" relation) and created new episodes.

In these episodes Jane, John, and Fred are people, Spot, Fido, and Rover are dogs, Felix is a cat, and Mort is a mouse. All of these are *objects* (also referred to as *entities*). Tokens of the same type are considered to be similar to each other, but not to tokens of other types – the example is simple enough not to require a hierarchy of types. 'Bite', 'flee', 'stroke', 'lick', and 'cause' are *relations*. *First-order* relations are those which have

Probe: Spot bit Jane, causing Jane to flee from Spot.

		Aspects of similarity						
Episodes in long-term memory:		OA	FOR	HOR	RFB	HOS	OLI	Type
E1:	Fido bit John, causing John to flee from Fido.	✓	✓	✓	✓	✓	✓	LS
E2:	Fred bit Rover, causing Rover to flee from Fred.	✓	✓	✓	×	✓	✓	AN^{cm}
E3:	Felix bit Mort, causing Mort to flee from Felix.	×	✓	✓	×	✓	✓	AN_1
E4:	Mort bit Felix, causing Felix to flee from Mort.	×	✓	✓	×	✓	✓	AN_2
E5:	Rover bit Fred, causing Rover to flee from Fred.	✓	✓	✓	$\frac{1}{2}$	✓	×	$SS^{\times I}$
E6:	John fled from Fido, causing Fido to bite John.	✓	✓	✓	✓	×	✓	$SS^{\times H}$
E7:	Mort bit Felix, causing Mort to flee from Felix.	×	✓	✓	×	✓	×	$FA^{\times I}$
E8:	Mort fled from Felix, causing Felix to bite Mort.	×	✓	✓	×	×	✓	$FA^{\times H}$
E9:	Fido bit John, John fled from Fido.	✓	✓	×	✓	×	✓	SS^{-H}
E10:	Fred stroked Rover, causing Rover to lick Fred.	✓	×	✓	×	×	×	OO_1
E11:	Fred stroked Rover, Rover licked Fred.	✓	×	×	×	×	×	OO_2

Table 6.3: The probe episode and the memorized episodes. Various aspects of similarity between each episode and the probe are marked with a check or a cross. The aspects of similarity stand for the following: OA: object attributes, FOR: first-order relations, RFB: role-filler bindings, HOR: higher-order relations, HOS: higher-order relational structure, OLI: object-level isomorphism (consistent mapping of objects). These are explained in Section 6.3.1 The types in the rightmost column are classifications of similarity, from Gentner *et al* [1993]: LS: Literal Similarity, AN: Analogy, SS: Surface Similarity, FA: False analogy, OO: Objects Only. These are explained in Section 6.3.2 The superscripts on the types are for later identification and indicate how the episode differs from the probe; ^{cm} indicates a cross-mapping (analogy), $^{\times I}$ indicates lack of object-level isomorphism, $^{-H}$ indicates missing higher-order relations, and $^{\times H}$ indicates mismatching higher-order structure.

only objects as arguments, i.e., 'bite', 'flee', 'stroke', and 'lick'. Relations that have other relations as arguments are *higher-order* relations. 'Cause' is the only higher-order relation in these examples. None of these relations are considered to be similar to any other. Each argument position of a relation corresponds to a *role* and the argument is the *filler* of the role. In "Fido bit John" (i.e., bite(Fido,John)), Fido is the filler of the *agent* role and John is the filler of the *object* role. Objects, relation names, and roles all can have *features* or *attributes*, e.g., is-a-person is a feature of Jane, John, and Fred.

This work does not tackle the problem of how visual or natural language input is converted into propositional form, which is a difficult problem in its own right. As with ARCS and MAC/FAC, the propositional form of the stories is the starting point. The propositional form of the episodes in Table 6.3 is straightforward, e.g., the propositions corresponding to the probe are:

```
p1  =  bite(Spot, Jane)
p2  =  flee(Jane, Spot)
p3  =  cause(p1, p2)
```

6.3.1 Aspects of similarity

I have identified six aspects of the similarities between the probe and the memorized episodes. The distinctions among these aspects relate to different treatment by SME or different effects on HRR dot-product similarity scores.

OA: Object attribute similarity – the presence of similar objects in each episode, e.g., people, dogs.

FOR: First-order relation similarity – the presence of similar first-order relations, e.g., bite, flee.

HOR: Higher-order relation similarity – the presence of similar higher-order relations, i.e., cause.

RFB: role-filler binding similarity – similar objects are bound to similar roles, e.g., a dog in the bite agent role. This only applies to bindings where the filler is an object, not another relation.

HOS: Higher-order relation structure similarity – similar relations are related in the same way, e.g., bite causes flee.

OLI: Object-level isomorphism – the presence of a consistent one-to-one mapping between the objects in the two episodes. This aspect is only important when objects fill multiple roles.

Object attribute similarity (OA), first-order relation similarity (FOR), and higher-order relation similarity (HOR) all depend on "surface" features – just the attributes of objects and relations. role-filler bindings similarity (RFB) depends on "local" structural features, while higher-order relation structure (HOS) and object-level isomorphism (OLI) depend upon the entire structural arrangement. I distinguish objects as fillers from predicates as fillers because the SME and ARCS treat objects differently to predicates.

Structural similarity, i.e., isomorphism, is split into two aspects – higher-order structure (HOS) and object-level isomorphism (OLI). It makes sense to treat the mapping of objects differently from the mapping of relations. The absence of similar objects does not at all diminish the strength of an analogy, and consequently object-level isomorphism does not require mapped objects to be similar. However, if two situations do not share similar relations, any analogy that can be constructed is likely to be abstract, difficult to find, and tenuous. Consequently, two episodes are similar in higher-order structure (HOS) if their higher-order structures are isomorphic and the corresponding predicates are similar. In SME objects mapped to each other can be dissimilar, but mapped relations must be identical. ARCS does allow mapping of non-identical relations, but only if they have the same number of arguments. The third structural aspect of similarity, role-filler binding (RFB) similarity is included because it has a large effect on HRR similarity but is ignored by SME and ARCS (although Markman *et al*[1993]) did note the stronger effect of "matches-in-place" versus "matches-out-of-place" on reminding and reasoning in human subjects).

In general, the rating of various aspects of similarity is best made on a continuous scale. However, the examples used here are so simple that a binary classification (✓ or ×) suffices. The exception is **E**5, which gets half a mark for role-filler binding similarity (RFB). The fillers of the two bite roles are similar to those of the probe (both have a dog biting a human), but the fillers of the two flee roles are not.

6.3.2 Classifications of similarity

The final column in Table 6.3 classifies the relationship between each episode and the probe using Gentner, Rattermann and Forbus's [1993] types of similarity, which have the following relationships to the presence of the various aspects of similarity:

LS (Literal Similarity): all aspects.

AN (Analogy, also called True Analogy): FOR, HOR, HOS, and OLI, but not OA or not RFB.

SS (Surface Similarity, also called Mere Appearance): OA, FOR, and HOR, but no structural aspects.

FA (False Analogy[4]): FOR and HOR only.

OO (Only Objects similarity): OA only.

E2 (AN^{cm}) is called a *cross-mapped* analogy because it involves the same types of objects as the probe, but the types of corresponding objects do not match – the types are switched around.

6.3.3 Ratings of the episodes

The first stages of MAC/FAC and ARCS only inspect surface features (OA, FOR, and HOR). The first stage of MAC/FAC (the "Many Are Called" stage) uses a vector representation of the surface features. Each location in the vector corresponds to a surface feature of an object, relation or function, and the value in the location is the number of times the feature occurs in the structure. The first-stage estimate of the similarity between two structures is the dot-product of their feature-count vectors. A threshold is used to select likely analogies. It would give **E**1 (LS), **E**2 (AN^{cm}), **E**5 ($SS^{\times I}$) and **E**6 ($SS^{\times H}$) equal and highest scores. The first stage of ARCS is less selective than that of MAC/FAC. It selects all episodes that share a relation with the probe, ignoring ubiquitous relations such as "cause" and "if". It would select all episodes involving "bite" or "flee", i.e., **E**1-**E**9.

ARCS and SME (the second stage of MAC/FAC) use very similar rules: mapped relations must match, all the arguments of mapped relations must be mapped consistently, and mapping of objects must be one-to-one. One minor difference is that SME requires exact matches of relations, while ARCS allows mapping of similar relations. This is only a minor difference because similar relations can be decomposed into identical and dissimilar components. SME has two modes – literal similarity mode, and analogy mode. In literal similarity mode, mappings between similar objects are favored over those between dissimilar objects, which gives literal similarity a higher score than analogy ("literal similarity" is analogy plus object attribute matches). Where it is relevant in this chapter, it is assumed that SME is operating in literal similarity mode.

The second stages of both MAC/FAC and ARCS would detect structural correspondences between each episode and the probe and give the literally similar and analogous episodes the highest rankings, i.e., LS > AN > (SS, FA, OO).

A simplified view of the overall similarity scores from SME and ARCS is shown in Table 6.4(b). The scores from the MAC similarity estimator (the first stage of MAC/FAC) are shown in Table 6.4(a). In the next three sections of this chapter I describe HRRs and how they can be used to compute fast similarity estimates that are more like ratings in Table 6.4(b).

[4]Gentner sometimes uses "First-Order Relational similarity" (FOR) instead of False analogy, but I avoid this term as it uses "First-order" in a different way (to refer to the consideration of relation names).

Structural Similarity	Object Attribute Similarity: YES	Object Attribute Similarity: NO
YES	(LS) High	(AN) Low
NO	(SS) High	(FA) Low

(a) Scores from the fast MAC similarity.

Structural Similarity	Object Attribute Similarity: YES	Object Attribute Similarity: NO
YES	(LS) High	(AN) ↕Med-High
NO	(SS) ↕Med-Low	(FA) Low

(b) Scores from a slow similarity estimator, e.g., SME or ARCS.

Table 6.4: A simplified view of the overall similarity scores from fast and slow similarity estimators. There are four conditions – the two structures being compared can be similar in structure and/or in object attributes. In all four conditions, the structures are assumed to involve similar relations – only structural and object attribute similarities are varied. Gentner and Forbus's names for these classes are shown in parentheses. Ideally, the responses to the mixed conditions should be flexible, and controlled by which aspects of similarity are currently considered important. Only their relative values of the scores are important, the absolute values do not matter.

6.4 Estimating similarity by dot-products of HRRs

6.4.1 Experiment 1

The dot-product of two HRRs gives an estimate of the overall similarity of the structures represented. This estimate combines the similarity on many different aspects, both surface and structural. The HRR dot-product is only an estimate of analogical match for two reasons:

1. The dot-product is noisy. The noise depends on essentially unpredictable interactions among the distributed representations of base vectors.

2. The expected value of the HRR dot-product is an imperfect measure of analogical similarity.

Experiment 1, described in this section, illustrates the ways in which the dot-product of ordinary HRRs reflects, and fails to reflect, the similarity of the underlying structures.

The set of base vectors and tokens used in Experiments 1, 2 and 3 is shown in Table 6.5.

The HRR for the probe is constructed as follows, and the HRRs for the other episodes are constructed in the same way.

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

Table 6.6 summarizes the dot-products between HRRs for the probe and the episodes **E**1-**E**11.

Experiment 1 was run 100 times, each time with a different choice of random base vectors. The vector dimension used was 2048. The means and standard deviations of the

Base vectors		Token vectors
person	**bite**	**jane** = $\langle \mathbf{person} + \mathbf{id}_{jane} \rangle$
dog	**flee**	**john** = $\langle \mathbf{person} + \mathbf{id}_{john} \rangle$
cat	**cause**	**fred** = $\langle \mathbf{person} + \mathbf{id}_{fred} \rangle$
mouse	**stroke**	**spot** = $\langle \mathbf{dog} + \mathbf{id}_{spot} \rangle$
	lick	**fido** = $\langle \mathbf{dog} + \mathbf{id}_{fido} \rangle$
$\mathbf{bite}_{agt}$	$\mathbf{bite}_{obj}$	**rover** = $\langle \mathbf{dog} + \mathbf{id}_{rover} \rangle$
$\mathbf{flee}_{agt}$	$\mathbf{flee}_{from}$	**felix** = $\langle \mathbf{cat} + \mathbf{id}_{felix} \rangle$
$\mathbf{cause}_{antc}$	$\mathbf{cause}_{cnsq}$	**mort** = $\langle \mathbf{mouse} + \mathbf{id}_{mort} \rangle$
$\mathbf{stroke}_{agt}$	$\mathbf{stroke}_{obj}$	
$\mathbf{lick}_{agt}$	$\mathbf{lick}_{obj}$	

Table 6.5: Base vectors and token vectors. All base and **id** vectors a randomly chosen with elements independently distributed as $N(0, 1/n)$.

P: Spot bit Jane, causing Jane to flee from Spot.

Episodes in long-term memory:		Aspects of similarity: OA	FOR	HOR	RFB	HOS	OLI	Type	Dot-products: Avg	Sd
E1:	Fido bit John, causing John to flee from Fido.	✓	✓	✓	✓	✓	✓	LS	0.70	0.016
E2:	Fred bit Rover, causing Rover to flee from Fred.	✓	✓	✓	×	✓	✓	AN^{cm}	0.47	0.022
E3:	Felix bit Mort, causing Mort to flee from Felix.	×	✓	✓	×	✓	✓	AN_1	0.39	0.024
E4:	Mort bit Felix, causing Felix to flee from Mort.	×	✓	✓	×	✓	✓	AN_2	0.39	0.024
E5:	Rover bit Fred, causing Rover to flee from Fred.	✓	✓	✓	$\frac{1}{2}$	✓	×	$SS^{\times I}$	0.58	0.019
E6:	John fled from Fido, causing Fido to bite John.	✓	✓	✓	✓	×	✓	$SS^{\times H}$	0.47	0.018
E7:	Mort bit Felix, causing Mort to flee from Felix.	×	✓	✓	×	✓	×	$FA^{\times I}$	0.39	0.024
E8:	Mort fled from Felix, causing Felix to bite Mort.	×	✓	✓	×	×	✓	$FA^{\times H}$	0.28	0.025
E9:	Fido bit John, John fled from Fido.	✓	✓	×	✓	×	✓	SS^{-H}	0.43	0.019
E10:	Fred stroked Rover, causing Rover to lick Fred.	✓	×	✓	×	×	×	OO_1	0.25	0.024
E11:	Fred stroked Rover, Rover licked Fred.	✓	×	×	×	×	×	OO_2	0.12	0.023

Table 6.6: Results from Experiment 1. The averages and standard deviations are for dot-products of the probe with each episode in memory, over 100 runs. The aspects of similarity required for analogy are first-order relation names (FOR), higher-order relation names (HOR), higher-order structure (HOS), and object-level isomorphism (OLI).

HRR dot-products of the probe and each episode are shown in Table 6.6. In 94 out of 100 runs, the ranking of the HRR dot-products was consistent with the following order (where the ordering within the parenthesis varies):

$$LS > SS^{\times I} > (AN^{cm}, SS^{\times H}) > (FA^{\times I}, AN_1, AN_2, SS^{-H}) > (FA^{\times H}, OO_1) > OO_2$$

This is also consistent with the order of the average dot-products. The order violations in individual runs are due to noise in the dot-products.[5] The variance of the noise decreases as the vector dimension increases. When the experiment was rerun with vector dimension 4096 there was only one violation of this order out of 100 runs.

These results illustrate two important properties of the HRR dot-product estimate of similarity. The first is that the HRR dot-product is more sensitive to structural similarity

[5] In each run of an experiment each dot-product has a fixed value, but the values are different with different choices of base vectors.

when the objects involved are similar. The second is that the HRR dot-product is very sensitive to having similar roles filled by similar entities.

6.4.2 Conditions for structural similarity to affect the HRR dot-product

The HRR dot-product (as in Experiment 1) will give a high score to an analogous pair of episodes if two conditions are satisfied:

1. The corresponding pairs of objects are similar, i.e., they are of the same or similar types.

2. The objects within each episode are of distinct types.

The episodes **E**1 (LS), **E**5 ($SS^{\times I}$) and **E**6 ($SS^{\times H}$) satisfy these two conditions with respect to the probe **P**, and the HRR dot-product ranks them correctly. Each of these episodes involves one dog and one person, as does the probe. The literally similar episode **E**1, which is analogous to the probe, receives a higher score than the two superficially similar episodes **E**5 and **E**6.

The requirement for corresponding pairs of objects to be similar is illustrated by the low score given to the cross-mapped analogy **E**2. Even though this episode is structurally isomorphic to the probe it does not score higher than the two superficially similar episodes **E**5 ($SS^{\times I}$) and **E**6 ($SS^{\times H}$).

The requirement of object attribute similarity for the detection of structural similarity is illustrated by the relative scores of **E**3 (AN_1) and **E**7 ($FA^{\times I}$). These two episodes do not share object attributes with the probe, and receive approximately equal scores despite their different degrees of structural similarity.

6.4.3 Why the HRR dot-product reflects structural similarity

The HRR for an episode can be expanded to a weighted sum of convolution products. This is illustrated for the probe in Figure 6.2. These convolution products contain one or more entities and correspond to the binding chains (hierarchical role-filler paths) in the episode. The weights come from normalizations performed during the construction of the HRR (and any relative component weights used). The binding chains provide a way of expressing structural features of an episode as superficial features of the HRR vector. Consideration of the binding chains shows both why the HRR dot-product reflects structural similarity under some conditions and why it is an unreliable indicator when those conditions are not satisfied. Two binding chains (convolution products) are similar if the entities in them are similar. Consequently, the HRRs for two episodes are similar if they have similar binding chains. Two episodes that are structurally similar and have similar fillers in similar roles will have similar binding chains. However, structurally similar episodes that do not have similar fillers in similar roles will not have similar binding chains (e.g., cross-mapped analogies). And episodes that have many similar fillers in similar roles but are structurally dissimilar may still have many similar binding chains (e.g., superficially similar episodes).

The vectors and binding chains which cause sensitivity to the six aspects of similarity are as follows:

OA (Object Attributes): Fillers in the top-level relation (e.g., $\mathbf{P}_{objects}$ in **P**).

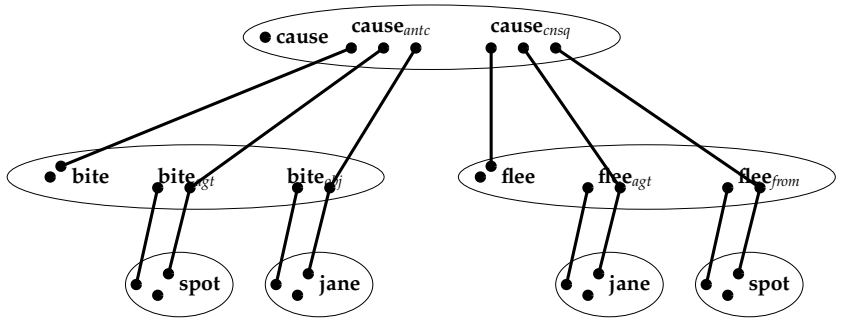


Figure 6.2: The HRR for the probe is the weighted sum of binding chains. Binding chains are the vertically connected dots. Weights are not shown. For example, the rightmost binding chain is **cause**$_{cnsq}$ ⊛ **flee**$_{from}$ ⊛ **spot**. Many roles and fillers participate in more than one binding chain.

FOR (First-order relation names): Relation names in the lower-level relations, and those relations in the top-level relation (e.g., **bite** in $\mathbf{P}_{bite}$ and **flee** in $\mathbf{P}_{flee}$, and $\mathbf{P}_{bite}$ and $\mathbf{P}_{flee}$ in **P**).

HOR (Higher-order relation names): Relation names in the higher-level relations (e.g., **cause** in **P**).

RFB (Role-filler bindings): First-order role-filler bindings, and those relations in the top-level relation (e.g., bindings in $\mathbf{P}_{bite}$ and $\mathbf{P}_{flee}$, and $\mathbf{P}_{bite}$ and $\mathbf{P}_{flee}$ in **P**).

HOS (Higher-order structure): Relation names in lower-level relations, and the role-filler bindings those relations participate in (e.g., **bite** in $\mathbf{P}_{bite}$ and **flee** in $\mathbf{P}_{flee}$, and the bindings **cause**$_{antc}$ ⊛ $\mathbf{P}_{bite}$ and **cause**$_{cnsq}$ ⊛ $\mathbf{P}_{flee}$ in **P**).

OLI (Object-level isomorphism): HRRs are not directly sensitive to this, though, in many situations OLI is correlated with role-filler binding similarity (RFB).

SME constrains corresponding higher-order relations to be identical. If this is a valid constraint, or even if just a high degree of similarity is required, then the detection of higher-order relational similarity is simpler than the detection of object-level isomorphism. Under this definition, higher-order relational structure is expressed as a superficial feature in a HRR vector – increasing structural similarity (while keeping the degree surface of surface similarity constant) will always increase the HRR dot-product. However, the object-level filler structure is not expressed as a superficial feature in the HRR vector – two episodes may have completely isomorphic object-level filler structure and low HRR dot-product similarity.

Compared to the similarity measures embodied in SME and ARCS, HRR dot-products are especially sensitive to the presence of similar role-filler bindings. In HRRs, binding similarity is the only mechanism by which structural similarity is detected. SME and ARCS use different (more rigorous) methods to detect structural isomorphism, and are more relaxed about binding similarity. However, they do take the similarity of mapped entities into account, and this is closely related to role-filler binding similarity. SME insists

that mapped relations should be identical (in name), but does not take account of the similarity of mapped objects unless it is operating in "Literal similarity" mode. ARCS has the soft constraint that all mapped entities, objects and relation names, should be similar.

A set of binding chains is not an unambiguous description of a structure – there are distinct structures that have identical sets of binding chains. The simplest examples of this are episodes with two instances of the same predicate, and no higher-order predicates. For example "Spot bit Jane, Fido bit Fred" has the same set of binding chains as "Spot bit Fred, Fido bit Jane".[6] If this turns out to be a problem, one could include filler-filler bindings (i.e., **spot** ⊛ **jane** for "Spot bit Jane") in order to better indicate which fillers belong to the same relation. The associativity and commutativity of convolution is another potential source of ambiguity. These properties result in the convolution product representation for binding chains being order-insensitive. This means that some binding chains arising from nested predicates are the same regardless of the nesting order of the predicates. If this turns out to be a serious problem, there are non-commutative versions of convolution which can be used (Section 3.6.7).

6.4.4 HRRs and multiple occurrences of objects

HRRs do not encode multiple occurrences of a single object as an explicit feature. Each occurrence is treated as a separate object – no links are made among multiple occurrences. This has several consequences for the HRR dot-product as a measure of analogical similarity. One is that HRRs are insensitive to object-level isomorphism (OLI) in the absence of object attribute similarity (OA). Another is that lack of object-level isomorphism is not detected if there are multiple objects of the same type. The HRR dot-product of the probe **P** and an episode will be high as long as the episode involves a dog biting a person, causing a person to run away from a dog. It does not matter whether the first person is the same as the second person, or the first dog the same as the second. For example, "Fido bit John, causing Fred to run away from Rover" would score just as highly as **E**1. HRRs dot-products are only sensitive to object-level isomorphism (OLI) through virtue of sensitivity to the presence of similar role-filler bindings (RFB). In Section 6.5, I describe how links can be made among the multiple occurrences of an object.

6.4.5 Weighting different aspects of the HRR

If certain aspects of the structures should have more influence than others on the HRR similarity scores, then these aspects can be given a higher relative weight in the HRR. For example, if it were desired that objects should have a very large influence on HRR similarity scores, the component $\mathbf{P}_{objects}$ could be scaled up before adding it to the other components of **P**. Changes in weighting only need to be done in one HRR of the comparison to produce an effect. This is computationally convenient – it means that the salience of different aspects of similarity can be changed by just reweighting components of the probe.

A version of Experiment 1 with $\mathbf{P}_{objects}$ given higher weight:

$$\mathbf{P}' = \langle \mathbf{cause} + 5\mathbf{P}_{objects} + \mathbf{P}_{bite} + \mathbf{P}_{flee} + \mathbf{cause}_{antc} \circledast \mathbf{P}_{bite} + \mathbf{cause}_{cnsq} \circledast \mathbf{P}_{flee} \rangle.$$

[6] If chunks for "Spot bit Jane" and "Fido bit Fred" are stored separately, these episodes can still be correctly decoded.

The HRR dot-product scores for this weighted version of the probe with the unweighted HRRs for each episode satisfied the following order, with one violation out of 100:

$$\mathrm{LS} > (\mathrm{SS}^{\times \mathrm{I}}, \mathrm{SS}^{-\mathrm{H}}, \mathrm{SS}^{\times \mathrm{H}}, \mathrm{AN}^{\mathrm{cm}}) > (\mathrm{OO}_1, \mathrm{OO}_2) > (\mathrm{AN}_1, \mathrm{AN}_2, \mathrm{FA}^{\times \mathrm{I}}) > \mathrm{FA}^{\times \mathrm{H}})$$

In this ranking the episodes which share object attributes with the probe score higher than those which do not.

In the HRRs described in this chapter, some relative weighting is accomplished by the normalizations. The components $\mathbf{P}_{bite}$, $\mathbf{P}_{flee}$, and $\mathbf{P}_{objects}$ are all normalized (by a factor of approximately $1/\sqrt{3}$, $1/\sqrt{3}$, and $1/\sqrt{2}$ respectively) so that they have equal weight in the final HRR. Thus the predicate names **bite** and **flee** have less weight in **P** than **cause**.

6.4.6 Covariance of HRR dot-products

If two episodes in memory are similar, their dot-products with the probe will tend to be correlated. This makes ordering of similarity more reliable than might be deduced by looking at the sample variances (shown in Table 6.6). For example, the difference between $\mathbf{P} \cdot \mathbf{E}1$ and $\mathbf{P} \cdot \mathbf{E}6$ (in each run) has an average value of 0.22 and a sample standard deviation of 0.017. If these two dot-products were uncorrelated the standard deviation of the difference would be around 0.024 (from summing the variances of $\mathbf{P} \cdot \mathbf{E}1$ and $\mathbf{P} \cdot \mathbf{E}6$). The correlation coefficient between $\mathbf{P} \cdot \mathbf{E}1$ and $\mathbf{P} \cdot \mathbf{E}6$ was 0.46 (which is significant beyond 0.2%). The correlation comes from common terms in the expansion of the dot-product, e.g., both **E**1 and **E**6 have **cause** + ⟨**john** + **fido**⟩ as a component, thus (**cause** + ⟨**john** + **fido**⟩) · (**cause** + ⟨**john** + **fido**⟩) contributes to both $\mathbf{P} \cdot \mathbf{E}1$ and $\mathbf{P} \cdot \mathbf{E}6$. This effect is much larger when base vectors and chunks are not normalized, but are chosen to have an expected length of one. The effect is larger because individual vectors contribute correlated values (e.g., **cause** · **cause**) to the final dot-products, whereas when all entities are normalized these contributed values are constant (because **cause** · **cause** = 1). An experiment without normalization yielded a correlation coefficient of 0.75 for $\mathbf{P} \cdot \mathbf{E}1$ and $\mathbf{P} \cdot \mathbf{E}6$. Standard deviations of these dot-products (and differences) were 50 to 150% higher than with normalization.

6.4.7 Properties of the HRR dot-product as a measure of similarity

Under certain conditions the HRR dot-product provides a good indication of overall similarity. However, it has two weaknesses:

1. It overestimates when structural similarity is low and there are many similar role-filler bindings.

2. It underestimates when structural similarity is high and there are few similar role-filler bindings.

These weaknesses are in part due to HRRs not encoding the multiple occurrence of a single entity as an explicit surface feature. The occurrence of a single entity in a multiple roles is what makes structural alignment difficult. If each entity occurs only once then the combinatoric complexity of the problem is much lower, and the problem is one of tree matching rather than graph matching.

This reliance on the presence of similar role-filler bindings as an indicator of structural similarity is the underlying reason for the failure of the HRR dot-product to detect structural similarity when corresponding fillers are not similar. The next section presents

an enhancement of HRRs intended to address this problem. This enhancement involves special treatment for objects which fill multiple roles and improves performance on cross-mapped analogies and on analogies without similar objects

6.5 Contextualized HRRs

Dot-product comparisons of HRRs are not sensitive to object-level isomorphism, especially in the absence of similarity among objects. This is because the way in which objects fill multiple roles is not expressed as a surface feature in HRRs. Consequently, the analogical episodes **E**2 (AN^{cm}), **E**3 (AN_1), and **E**4 (AN_2) are given lower scores than the superficially similar episode **E**5 ($SS^{\times I}$).

We can force role structure to become a surface feature (and thus allow the degree of role alignment to influence the dot-product comparison) by "contextualizing" the representations of fillers. Contextualization involves incorporating information about what other roles an object fills in the representation of a filler. This is like thinking of Spot (in the probe) as an entity which bites (a biter) and an entity which is fled from (a "fledfrom"). Contextualization can be implemented in HRRs by blending the representation for Spot with a representation for the typical fillers of the other roles Spot fills.

One way to represent a typical bite agent filler (a biter) is:

$$\mathbf{bite}_{typagt} = \mathbf{bite} * \mathbf{bite}^{*}_{agt},$$

where $\mathbf{bite}^{*}_{agt}$ is the approximate inverse of $\mathbf{bite}_{agt}$. This representation for $\mathbf{bite}_{typagt}$ is attractive because it is a component of the result of decoding the agent of any "bite" predicate. The agent of a predicate is decoded by convolving with the approximate inverse of the appropriate agent vector as follows:

$$\begin{aligned}(\mathbf{bite} + \mathbf{bite}_{agt} \circledast \mathbf{spot} + \mathbf{bite}_{obj} \circledast \mathbf{jane}) \circledast \mathbf{bite}^{*}_{agt} \\ \approx \quad \mathbf{spot} + \underline{\mathbf{bite} \circledast \mathbf{bite}^{*}_{agt}} + \mathbf{bite}_{obj} \circledast \mathbf{jane} \circledast \mathbf{bite}^{*}_{agt}\end{aligned}$$

The underlined term is always present when an agent is extracted from a bite predicate, but would usually be treated as noise. The attractiveness of this representation for typical fillers is in its potential for developing representations of entities that contain information about the types of roles and relationships the entity is commonly involved in. If the long-term memory representation for Spot is somewhat malleable, and if Spot is decoded as the agent in many bite predicates, then the representation for Spot might over time incorporate this component which represents a typical biter. However, the implementation of this is outside the scope of this thesis.

In ordinary HRRs the filler alone is convolved with the role. In contextualized HRRs a blend of the filler and its context is convolved with the role. The representation for the context of object in a role is the typical fillers of the *other* roles the object fills. The context for Spot in the **flee** relation is represented by $\mathbf{bite}_{typagt}$ and the context in the **bite** relation is represented by $\mathbf{flee}_{typfrom}$. The degree of contextualization is governed by the mixing proportion κ_o (object) and κ_c (context). The contextualized HRR for the probe is constructed as follows:

$$\begin{aligned}\mathbf{P}_{bite} &= \langle \mathbf{bite} + \mathbf{bite}_{agt} \circledast (\kappa_o \mathbf{spot} + \kappa_c \mathbf{flee}_{typfrom}) \\ &\quad + \mathbf{bite}_{obj} \circledast (\kappa_o \mathbf{jane} + \kappa_c \mathbf{flee}_{typagt}) \rangle \\ \mathbf{P}_{flee} &= \langle \mathbf{flee} + \mathbf{flee}_{agt} \circledast (\kappa_o \mathbf{jane} + \kappa_c \mathbf{bite}_{typobj}) \\ &\quad + \mathbf{flee}_{from} \circledast (\kappa_o \mathbf{spot} + \kappa_c \mathbf{bite}_{typagt}) \rangle \\ \mathbf{P}_{objects} &= \langle \mathbf{jane} + \mathbf{spot} \rangle \\ \mathbf{P}_{probe} &= \langle \mathbf{cause} + \mathbf{P}_{objects} + \mathbf{P}_{bite} + \mathbf{P}_{flee} + \mathbf{cause}_{antc} \circledast \mathbf{P}_{bite} + \mathbf{cause}_{cnsq} \circledast \mathbf{P}_{flee} \rangle\end{aligned}$$

The degree of contextualization controls the degree to which the pattern of fillers is expressed as a surface feature, and hence the controls the degree to which role alignment influences the dot-product.

6.5.1 Flexible salience of object-level isomorphism

Human judgements of similarity are very flexible – the salience of different aspects of similarity can be changed by context or command. The degree to which role alignment affects the HRR dot-product can be adjusted by changing the degree of contextualization in just one episode of a pair. Hence, the items in memory can be encoded with a fixed κ values (κ^m_o and κ^m_c) and the salience of role alignment can be changed by altering the degree of contextualization in the probe (κ^p_o and κ^p_c). This is fortunate as it would be impractical to recode all items in memory in order to alter the salience of role alignment in a particular comparison.

6.6 Estimating similarity by dot-products of contextualized HRRs

Two experiments were performed with contextualized HRRs, with the same episodes as used in Experiment 1. In Experiment 2 the probe was non-contextualized ($\kappa^p_o = 1, \kappa^p_c = 0$), and in Experiment 3 contextualization was used ($\kappa^p_o = 1/\sqrt{2}, \kappa^p_c = 1/\sqrt{2}$). The episodes in memory were encoded with the same degree of contextualization ($\kappa^m_o = 1/\sqrt{2}, \kappa^m_c = 1/\sqrt{2}$) for both experiments. As before, each set of comparisons was run 100 times, and the vector dimension was 2048. The results are listed in Table 6.7. Only the means of the dot-products are shown – the standard deviations were very similar to those in Experiment 1.

6.6.1 Experiments 2 and 3

The dot-products in Experiment 2 were consistent (in 95 out of 100 runs) with the same order as given for Experiment 1:

$$\mathrm{LS} > \mathrm{SS}^{\times \mathrm{I}} > (\mathrm{AN}^{\mathrm{cm}}, \mathrm{SS}^{\times \mathrm{H}}) > (\mathrm{FA}^{\times \mathrm{I}}, \mathrm{AN}_1, \mathrm{AN}_2, \mathrm{SS}^{-\mathrm{H}}) > (\mathrm{FA}^{\times \mathrm{H}}, \mathrm{OO}_1) > \mathrm{OO}_2$$

The dot-products in Experiment 3 were consistent (in all 100 runs) with an ordering which ranks analogical episodes as strictly more similar than non-analogical ones:

$$\mathrm{LS} > \mathrm{AN}^{\mathrm{cm}} > (\mathrm{AN}_2, \mathrm{AN}_1) > (\mathrm{SS}^{\times \mathrm{I}}, \mathrm{SS}^{\times \mathrm{H}}, \mathrm{SS}^{-\mathrm{H}}) > (\mathrm{FA}^{\times \mathrm{I}}, \mathrm{FA}^{\times \mathrm{H}}) > \mathrm{OO}_1 > \mathrm{OO}_2$$

6.6.2 Discussion of results

When there is no contextualization in probe ($\kappa^p_o = 1, \kappa^p_c = 0$) the ordering of dot-products is the same as for Experiment 1. With contextualization, the analogous episodes **E**2 (AN^{cm}),

P: Spot bit Jane, causing Jane to flee from Spot.

	Episodes in long-term memory:	Aspects of similarity OA	FOR	HOR	RFB	HOS	OLI	Type	Avg. dot-prods Expt2	Expt3
E1:	Fido bit John, causing John to flee from Fido.	✓	✓	✓	✓	✓	✓	LS	0.63	0.81
E2:	Fred bit Rover, causing Rover to flee from Fred.	✓	✓	✓	×	✓	✓	AN^{cm}	0.47	0.69
E3:	Felix bit Mort, causing Mort to flee from Felix.	×	✓	✓	×	✓	✓	AN_1	0.39	0.61
E4:	Mort bit Felix, causing Felix to flee from Mort.	×	✓	✓	×	✓	✓	AN_2	0.39	0.61
E5:	Rover bit Fred, causing Rover to flee from Fred.	✓	✓	✓	$\frac{1}{2}$	✓	×	$SS^{\times I}$	0.55	0.53
E6:	John fled from Fido, causing Fido to bite John.	✓	✓	✓	✓	×	✓	$SS^{\times H}$	0.44	0.53
E7:	Mort bit Felix, causing Mort to flee from Felix.	×	✓	✓	×	✓	×	$FA^{\times I}$	0.39	0.39
E8:	Mort fled from Felix, causing Felix to bite Mort.	×	✓	✓	×	×	✓	$FA^{\times H}$	0.27	0.39
E9:	Fido bit John, John fled from Fido.	✓	✓	×	✓	×	✓	SS^{-H}	0.38	0.51
E10:	Fred stroked Rover, causing Rover to lick Fred.	✓	×	✓	×	×	×	OO_1	0.25	0.25
E11:	Fred stroked Rover, Rover licked Fred.	✓	×	×	×	×	×	OO_2	0.12	0.12

Table 6.7: Results from Experiments 2 and 3. Average dot-products of non-contextualized probe (Experiment 2) and contextualized probe (Experiment 3) with contextualized episodes in memory.

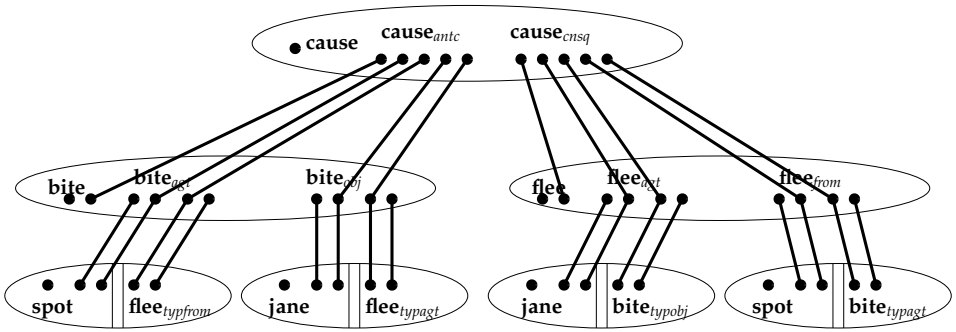


Figure 6.3: The binding chains in the contextualized HRR for the probe. Compare with Figure 6.2. The incorporation of context almost doubles the number of binding chains.

E3 (AN_1), and **E**4 (AN_2) receive a higher score than the superficially similar episode **E**5 ($SS^{\times I}$).

The example here shows the use of contextualization for objects, but it can also be used for relations. The example is not complex enough to have a relation appearing in multiple roles. For example, if "Jane fled from Spot" filled a role in another relation in addition to filling the cause antecedent role, then **cause**$_{typcnsq}$ would be the context used in that other relation.

6.6.3 Extra binding chains introduced by contextualization

Contextualization nearly doubles the number of binding chains in the HRR for an episode (and would increase their number more if some entities were involved in more than two relations). Figure 6.3 shows the binding chains in the contextualized HRR for the probe. The extra binding chains encode the patterns in which objects fill multiple roles.

6.6.4 Limits of contextualization

Contextualization does *not* cause all relational structure to be expressed as surface features in the HRR vector. It only suffices to distinguish analogical from non-analogical structures when no two entities fill the same set of roles. Sometimes, the distinguishing context for an object is more than the other roles that the object fills. Consider the situation where two boys are bitten by two dogs, and each flees from the dog which did not bite him. With contextualization as described above it is impossible to distinguish this from the situation where each boy flees from the dog that did bite him. The point is that the structural context required to distinguish objects is more than the set of roles the objects fill. This example with two boys and two dogs can be written as follows (where p is the bite relation and q is the flee relation):

$$\begin{aligned} S_1 &= p(x,y), p(w,z), q(z,x), q(y,w) \\ S_2 &= p(a,b), p(c,d), q(d,a), q(b,c) \\ S_3 &= p(a,b), p(c,d), q(b,a), q(d,c) \end{aligned}$$

S_2 is analogous to S_1, but S_3 is not. However, if objects are ignored, they all have the same contextualized representation (where tf_{q_1} is the typical filler of the first role of q etc):

$$p(tf_{q_2}, tf_{q_1}), p(tf_{q_2}, tf_{q_1}), q(tf_{p_2}, tf_{p_1}), q(tf_{p_2}, tf_{p_1})$$

One could try to get around these limits on contextualization by incorporating more information into filler contexts, but the scheme quickly loses its simplicity. Problems with circular definitions arise – if other fillers are taken into account then the contextualized representation of x in $p(x,y)$ involves the contextualized representation of y, which involves the contextualized representation of x. The solution probably lies in performing deeper contextualization only where necessary, for example, contextualizing John as "the person who fled from the dog which didn't bite him".

What is important is whether these limitations of contextualization ever manifest themselves in practical problems. Contextualization is more powerful than it appears at first sight – it fails only when there is a high degree of symmetry within as well as between episodes.

6.7 Interpretations of an analogy

To accurately evaluate the goodness of an analogy, or to use an analogy to help with reasoning, we need to work out which entities correspond to each other. The dot-product of HRRs does not provide this. However, corresponding entities can be easily computed from HRRs, by decoding the appropriate roles and fillers using the same techniques as described in Section 3.10.3.

Consider the probe **P** "Spot bit Jane, causing Jane to flee from Spot", and **E**1 "Fido bit John, causing John to flee from Fido." The entity corresponding to Jane (i.e., John) can be found in two steps:

1. Extract the roles Jane fills from the probe with the operation: $\langle \mathbf{P} \circledast \mathbf{jane}^* \rangle$ The roles which have positive dot-products with this result are:

$\langle \mathbf{E1} \circledast (\mathbf{P} \circledast \mathbf{jane}^*)^* \rangle$	**john**	0.37	✓
	fido	0.05	
$\langle \mathbf{E2} \circledast (\mathbf{P} \circledast \mathbf{jane}^*)^* \rangle$	**fred**	0.25	×
	rover	0.10	
$\langle \mathbf{E3} \circledast (\mathbf{P} \circledast \mathbf{jane}^*)^* \rangle$	**mort**	0.14	✓
	felix	0.02	
$\langle \mathbf{E4} \circledast (\mathbf{P} \circledast \mathbf{jane}^*)^* \rangle$	**felix**	0.12	✓
	mort	0.07	
$\langle \mathbf{E5} \circledast (\mathbf{P} \circledast \mathbf{jane}^*)^* \rangle$	**fred**	0.26	✓
	rover	0.08	

(a)

$\langle \mathbf{E1} \circledast \mathbf{R}^* \rangle$	**john**	0.22	✓
	fido	0.11	
$\langle \mathbf{E2} \circledast \mathbf{R}^* \rangle$	**rover**	0.25	✓
	fred	0.08	
$\langle \mathbf{E3} \circledast \mathbf{R}^* \rangle$	**mort**	0.24	✓
	felix	0.10	
$\langle \mathbf{E4} \circledast \mathbf{R}^* \rangle$	**felix**	0.25	✓
	mort	0.10	
$\langle \mathbf{E5} \circledast \mathbf{R}^* \rangle$	**fred**	0.21	✓
	rover	0.17	

(b)

Table 6.8: Interpretation of an analogy: extraction of entities from other episodes which correspond to Jane in **P**. (a) shows results without intermediate clean-up. (b) shows the results using intermediate role clean-up (**R** is specified in the text). Correct extractions are checkmarked. Intermediate role clean-up is necessary to correctly extract the corresponding entity from **E**2 (AN^{cm}).

$\langle \mathbf{P} \circledast \mathbf{jane}^* \rangle$	$\mathbf{flee}_{agt}$	0.17
	$\mathbf{cause}_{antc}$	0.16
	$\mathbf{bite}_{agt}$	0.13
	$\mathbf{cause}_{cnsq}$	0.10
	$\mathbf{stroke}_{agt}$	0.01

2. Extract the fillers of those roles from **E**1 and compare (dot-product) with the entities in **E**1:

$\langle \mathbf{E1} \circledast (\mathbf{P} \circledast \mathbf{jane}^*)^* \rangle$	**john**	0.37
	fido	0.05

The most similar entity is John, which is the entity in **E**1 corresponding to Jane.

The extraction of the entities corresponding to Jane in $\mathbf{E1} - \mathbf{E5}$ is shown in Table 6.8(a). The correct answer is obtained in **E**1 (LS), where corresponding objects are similar, and in **E**3 (AN_1) and **E**4 (AN_2), where there is no object similarity. This extraction process has a bias towards choosing similar entities as the corresponding ones. There is no correct answer for **E**5 ($SS^{\times 1}$), because there is no consistent mapping between **P** and **E**5. However, Fred, (the entity more similar to Jane) is strongly indicated to be the one corresponding to Jane. The wrong answer is given for the cross-mapped analogy **E**2, where again the more similar object is indicated to be the corresponding one.

Closer examination of the extraction process reveals both the reason for this bias and a way of eliminating it. Consider the expansion of $\mathbf{P} \circledast \mathbf{jane}^*$ (omitting weighting factors):

$$\mathbf{P} \circledast \mathbf{jane}^* = \ldots + \mathbf{bite}_{obj} + \ldots + \mathbf{flee}_{agt} + \ldots + \mathbf{cause} \circledast \mathbf{jane}^* + \ldots$$

The terms $\mathbf{bite}_{obj}$ and $\mathbf{flee}_{agt}$ are what we want – they come from $\mathbf{bite}_{obj} \circledast \mathbf{jane} \circledast \mathbf{jane}^*$ and $\mathbf{flee}_{agt} \circledast \mathbf{jane} \circledast \mathbf{jane}^*$). However, the other terms like $\mathbf{cause} \circledast \mathbf{jane}^*$ are the source of the bias. The approximate inverse[7] of $\mathbf{P} \circledast \mathbf{jane}^*$ is:

$$(\mathbf{P} \circledast \mathbf{jane}^*)^* = \ldots + \mathbf{bite}^*_{obj} + \ldots + \mathbf{flee}^*_{agt} + \ldots + \mathbf{cause}^* \circledast \mathbf{jane} + \ldots$$

When this is used to extract the fillers from **E**2,

$$(\mathbf{P} \circledast \mathbf{jane}^*)^* \circledast \mathbf{E2},$$

the result includes **fido** from $\mathbf{bite}_{obj} \circledast \mathbf{fido} \circledast \mathbf{bite}^*_{obj}$ and $\mathbf{flee}_{agt} \circledast \mathbf{fido} \circledast \mathbf{flee}^*_{agt}$. The result also includes a large component of **jane** from the many terms like $\mathbf{cause} \circledast \mathbf{cause}^* \circledast \mathbf{jane}$. The **jane** component is much larger than the **fido** component, and as **jane** is similar to **fred** the result ends up being more similar to **fred** than **fido**.

This bias can be eliminated by performing a special intermediate clean-up on the roles extracted by the operation $\mathbf{P} \circledast \mathbf{jane}^*$. This more complex clean-up should pass all positive role components and suppress negative role and non-role components like $\mathbf{cause} \circledast \mathbf{jane}^*$. For example, the cleaned-up version of $\mathbf{P} \circledast \mathbf{jane}^*$ should be:

$$\mathbf{R} = (0.17 * \mathbf{flee}_{agt} + 0.15 * \mathbf{bite}_{obj} + 0.13 * \mathbf{cause}_{antc} + 0.10 * \mathbf{cause}_{cnsq} + 0.01 * \mathbf{stroke}_{agt}),$$

where the weights come from the dot-products of $\mathbf{P} \circledast \mathbf{jane}^*$ and the roles (listed in step 1 at the beginning of this section). This clean-up can be viewed as a less competitive version of the standard clean-up. The corresponding objects extracted using role clean-up are shown in Table 6.8(b). The process gives the correct answers for all four episodes where there is a consistent mapping, and gives an ambiguous answer for the episode which has no consistent mapping with the probe. The process is not sensitive to the weights used in the clean-up – similar answers are obtained if the roles are added together without weights (as long as just those roles which have a positive dot-product with $\mathbf{P} \circledast \mathbf{jane}^*$ are used), i.e,

$$\mathbf{R} = (\mathbf{flee}_{agt} + \mathbf{bite}_{obj} + \mathbf{cause}_{antc} + \mathbf{cause}_{cnsq} + \mathbf{stroke}_{agt}).$$

There are two problems with these techniques for deriving interpretations. One is that each pair in the mapping is extracted independently. This matters when there is more than one consistent mapping. For example, if we have two possible mappings $\{X \leftrightarrow A, Y \leftrightarrow B\}$ and $\{X \leftrightarrow B, Y \leftrightarrow A\}$, then the choice of mapping for X should constrain the choice for Y. Another problem is that these techniques are unsatisfactory when two different objects have the same set of roles – ambiguous results can be produced.

6.8 Discussion

6.8.1 The criteria for good analogies in SME and ARCS

The criteria for good analogies in SME and ARCS are very similar and are listed in Table 6.9. SME and ARCS are intended to model analogical reasoning in people, so these criteria come from observations, experiments, and theories about how people judge and reason with analogies.

The degrees to which dot-products of ordinary and contextualized HRRs can be seen as implementing the criteria in Table 6.9 are as follows:

[7] The approximate inverse obeys the axioms $(a + b)^* = a^* + b^*$ and $(a \circledast b^*)^* = a^* \circledast b$.

SME	ARCS
Clarity: it is clear which objects in the two situations correspond to each other.	**Semantic Similarity:** Mapped elements should be similar.
Richness: analogies with more correspondences are better.	**Isomorphism:** Consistent mappings are required: • Mappings should be one-to-one. • Structural consistency – if a relation is mapped, then its arguments should be mapped consistently as well.
Abstractness: correspondences among abstract entities are better: • Attributes (one-place predicates) are not mapped (unless operating in literal similarity mode). • Higher-order relations are more important.	**Pragmatic centrality:** Either a particular correspondence holds or a particular element is involved in some correspondence.
Systematicity: consistent mappings are required. This involves: • Presence of only one-to-one mappings. • Structural consistency – a relation can only be in a mapping only if its arguments are consistently mapped as well.	

Table 6.9: Criteria for good analogies in SME and ARCS.

Clarity: No. However, an indication of clarity can be obtained by computing corresponding objects and seeing how clear the choices are.

Richness: Yes. More similar components means a higher dot-product.

Abstractness: Not with ordinary HRRs, to some degree with contextualized HRRs. Having similar attributes on corresponding objects does contribute to dot-product similarity, but can be made less important by using contextualization. Higher-order relations can be given more importance by using them larger weights for them.

Semantic similarity: Yes. Semantic similarity between corresponding entities increases the HRR dot-product. (This is the opposite of abstractness.)

Systematicity (isomorphism):

- Structural consistency: To some degree. Two instances of a higher-order relation will be more similar if their arguments are similar.
- One-to-one mappings: Not with ordinary HRRs, to some extent with contextualized HRRs.

Pragmatic centrality: No. However, if it were important that a particular object or relation be matched, it could be given a higher weight.

6.8.2 Flexibility of comparisons

Gentner and Markman [1993] and Goldstone, Medin and Gentner [1991] present convincing evidence that people are flexible in how they evaluate or use analogies. People can take different aspects of situations into account, depending on what is salient in the given context. In the HRR model all aspects are added together but flexibility is retained – different aspects in the representations can be weighted as appropriate. This requires only changing the representation of the probe; the representations of items in long-term memory can be fixed.

6.8.3 Features or predicates?

In HRRs, features and predicates have different representations. This creates the problem of having to decide which to use. The choice can dramatically affect the similarity of ordinary HRRs, but has a lesser effect on the similarity of contextualized HRRs.

In their commentary on the shape configuration experiments, Markman *et al* suggest Items 5 and 6 demonstrate that multiple relations can be used in the same comparisons ("small object above large object" is more similar to "small object above large object" than "large object above small object"). In the HRR encoding I used features rather than relations to represent the relative sizes, and this gave acceptable results. It would be possible to encode the relative sizes as a predicate "is larger than" (with roles "larger" and "smaller"), rather than as features of the objects. However, if this were done the dot-product comparisons of ordinary HRRs would not produce the correct answers for Item 6, because there would be no explicit relationship (binding) between "smaller" and "above". Using contextualization and a "is larger than" predicate would have a very similar effect to using the features "large" and "small", because contextualization would provide a "typical larger object" feature and a "typical smaller object" feature to be added to the shape features (circle, star, etc). This suggests that there is something right about contextualization – if it lessens the artificial distinction between features and predicates

This issue is related to issues that arise with SME and ARCS – what should the arity of a predicate be, and should single-place predicates (i.e., features or attributes) match? SME and ARCS represent attributes (i.e., features) as single-place predicates,[8] so one does not have to choose between representing something as an attribute or a predicate. However, the number of arguments of a predicate must be chosen, and the choice does have consequences. An attribute or feature like "small" can be represented as a one- or two-place predicate, e.g., as `is-small(mouse)` or `is-smaller-than(mouse, elephant)`. The choice is important for two reasons: neither SME nor ARCS allows matches between predicates with different numbers of arguments, and SME (in analogy mode) does not allow matches between single-place predicates, unless they are part of some larger structure. This is a general problem because a multi-place predicate can always be turned into a single-place predicate by moving arguments into the predicate name, e.g., `fed-peanuts-to-the-monkeys-on-tuesday(Jill)`.

6.8.4 Inadequacy of vector dot-product models of similarity

Tversky [1977] pointed out the now-well-known reasons why human similarity judgements cannot be modelled as a simple vector dot-product. The problems arise because there

[8] Though ARCS's similarity matrix is an alternative location for representing attributes.

cases in which people's similarity judgements violate each of three axioms of metric spaces (with $d(\cdot,\cdot)$ as the distance function): minimality, $d(x,x) = 0$; symmetry, $d(x,y) = d(y,x)$; and the triangle inequality, $d(x,z) \leq d(x,y) + d(y,z)$.

However, these problems do not preclude the use of vector dot-products at some level in a model of similarity. Much of the problematic data could be explained by a vector dot-product model which allowed vectors to be transformed before computing the dot-product. The transform would depend on the context of the comparison and possibly on the items to be compared. It could be something as simple as an adjustment of the weighting of various components in the vector.

6.8.5 Scaling to have more episodes in memory

In the experiments reported here, the clean-up memory held only 12 episodes. However, many more episodes could be added without any decrease in accuracy of retrieval, provided the new episodes were not very similar to the existing ones. Retrieval or rating errors are most likely to occur with similar episodes; errors are extremely unlikely with non-similar episodes because the tails of a Gaussian distribution drop off so quickly. Thus, as long as any one episode is similar to only a small number of others, the system with $n = 2048$ or $n = 4096$ will be able to store many thousands of episodes.

6.8.6 Chunking, scaling, and networks of HRRs

One problem with the HRR dot-product as a measure of similarity is that it does not distinguish between a poorly matching pair of components and a well-matching but low-weighted pair of components. A possible solution to this could be to build a spreading activation network of chunks. Each episode would be represented by a collection of chunks, and each object and relation, and possibly even features, would be a separate chunk. For example, the chunks for "small-circle above large-square" could be as follows:

$$\mathbf{small}, \mathbf{circle}, \mathbf{smCircle}, \mathbf{large}, \mathbf{square}, \mathbf{lgSquare}, \mathbf{vertical},$$
$$\langle \mathbf{vertical} + \mathbf{above} \circledast \mathbf{smCircle} + \mathbf{below} \circledast \mathbf{lgSquare} \rangle, \text{ and}$$
$$\langle \langle \mathbf{smCircle} + \mathbf{lgSquare} \rangle + \langle \mathbf{vertical} + \mathbf{above} \circledast \mathbf{smCircle} + \mathbf{below} \circledast \mathbf{lgSquare} \rangle \rangle.$$

To build a network which matched this structure against another, we would need nodes for these chunks and those in the other structure, and we would build links between nodes whose strength was proportional to the dot-product similarity of the corresponding chunks. Some form of competition would be necessary to prevent all nodes going to maximum activation. The initial activation of chunks could reflect their a priori importance or salience. The network would be run by allowing it to settle to a stable state. The final state of the network would show which components of the structure were similar, and the degree-of-match between components could be easily extracted.

Another problem will probably occur when the HRR dot-product is used to estimate the similarity of episodes with many predicates. As recognized in SME and ARCS, a good analogy can occur when just one coherent part of one structure matches a part of the other. Thus, for larger episodes, a linear sum of matches of structural features is probably not a very appropriate measure of similarity. The dot-product measure does not distinguish between a large number of medium similarity matches and a small number of high similarity matches. Furthermore, while there is some sensitivity to the matches being localized in a part of the structure (due to the hierarchical binding chains), this effect is probably not very strong. Chunking, and using a spreading activation network of chunks is also a possible solution for this scaling problem. Large episodes would be stored as a number of chunks in long-term memory, and the chunks belong to the same episode would be quite similar. The probe episode would also consist of several chunks. The search for analogies to the probe in long-term memory would now consist of several steps. In the first step chunks that were similar to those of the probe would be retrieved. Next, chunks similar to retrieved chunks would be retrieved, in order to pull in entire (or large parts of) episodes from long-term memory. Next, a spreading activation network would be constructed as described above, and activation allowed to settle. The episode with the largest amount of activation would be considered the most similar. This process would still take structural similarity into account because chunks would not be similar unless the structures they came from were similar. It would also allow the weighting of various aspects (i.e., chunks) of each structure to change during and in response to the matching process. This would go a long way towards creating flexible connectionist model of similarity assessment like that speculated about by Gentner and Markman [1993].

6.8.7 Comparison to RAAMs and tensor-product models

Pollack's [1990] RAAMs are a type of backpropagation network which can learn to encode recursive structures in a fixed-width vector. Conceivably, one could get estimates of structural similarity by comparing the reduced representations discovered by RAAMs. Pollack shows some cluster diagrams which seem to have similar structures grouped together. However, the strength of this effect is unclear, and I am not aware of any systematic studies of this effect. Extrapolating from other things known about RAAMs, we could expect them to have two disadvantages relative to HRRs as a way of getting estimates of structural similarity:

1. RAAMs require much learning time and the generalization appears weak – there is no guarantee that a new structure can be represented at all (without extensive training). For example, if a RAAM were trained on episodes involving the predicates cause, bite, and flee, it might not even be able to represent episodes involving stroke or lick, let alone compare them to other episodes.

2. The non-linear nature of RAAMs makes it possible for similar structures to have quite different representations, even when the structures are similar in both surface and structural features.

On the other hand, the learning involved in RAAMs offers a potential advantage over HRRs – RAAMs could learn to devote more representational resources to commonly encountered structures and thus achieve better performance on them. The downside of this would be poorer performance on rarely encountered structures.

Smolenksy's [1990] tensor-product representations are another method for encoding hierarchical structure in distributed representations. They have much in common with HRRs, and it would not be difficult to use them to replicate the results in this chapter. The main difference between the two methods is that the dimensionality of the tensor-product representation increases exponentially with the depth of the structure, while the dimensionality of HRRs remains constant. This could lead to unreasonable resource requirements for structures with several levels of nesting. An advantage of tensor product representations is that they tend to be less noisy than HRRs in decoding, and would probably also give less noisy similarity estimates.

Structural Similarity	Object Attribute Similarity	
	YES	NO
YES	(LS) High	(AN) Low
NO	(SS) Med	(FA) Low

(a) Ordinary-HRR dot-products.

Structural Similarity	Object Attribute Similarity	
	YES	NO
YES	(LS) High	(AN) ↕Med-High
NO	(SS) ↕Med-Low	(FA) Low

(b) Contextualized-HRR dot-product.

Table 6.10: Approximate scores from ordinary and contextualized HRR dot-product comparisons. The flexibility comes adjusting the weights of various components in the probe.

The results here would be difficult to replicate with Halford *et al*'s [to appear] representation for relations (Section 2.4.3). Their representation for a relation is the tensor product of its arguments and name. For example, the relation `bite(Spot, Jane)` would be represented as **bite** ⊗ **spot** ⊗ **jane**. One problem with this representation is that relations with just one dissimilar argument are dissimilar, where dissimilar means zero similarity. For example, **bite** ⊗ **spot** ⊗ **jane** is dissimilar to **bite** ⊗ **spot** ⊗ **fido** (assuming **jane** and **fido** are dissimilar). Another problem is that it is not clear how to represent episodes with multiple or nested predicates. Halford *et al* do mention chunking, but do not give any details how it might be done. On the other hand, the computations Halford *et al* perform are easy to do with a role-filler binding representation (using either tensor products or convolution), provided that relations are stored in a content-addressable item memory i.e., a clean-up memory.

6.9 Conclusion

While it would be overstating the importance of this work to describe it as a watershed for connectionist modelling, it does adds further evidence that complex structure and structural alignment is not the Waterloo for connectionist techniques.

The dot-product of HRRs provides a fast estimate of the degree of analogical similarity and is sensitive to various structural aspects of the match. It is not intended to be a model of complex or creative analogy making, but it could be a useful first stage in a model of analogical reminding. The HRR dot-product is only an estimate of analogical similarity for two reasons: its actual value is noisy, although the noise decreases with increasing vector dimension, and its expected value is an imperfect measure of analogical similarity.

The dot-product of ordinary HRRs is sensitive to some aspects of analogical similarity. It improves on the existing fast similarity matcher in MAC/FAC in that it discriminates the first column of Table 6.4(a) – it ranks literally similar (LS) episodes higher than superficially similar (SS) episodes. However, it is insensitive to object-level isomorphism when corresponding objects are not similar. Consequently, it ranks both analogies (AN) and false analogies (FA) lower than superficially similar (SS) episodes. The approximate values of the estimates it gives are shown in Table 6.10(a) – compare these with Table 6.4(a).

The dot-product of contextualized HRRs is sensitive to more aspects of structural similarity. In particular, it is sensitive to object-level isomorphism even when corresponding objects are not similar. It ranks the given examples in the same order as would SME or ARCS. The approximate values of its estimates are shown in Table 6.10(b) – these are the same as the SME and ARCS scores in Table 6.4(b).

HRR dot-products are flexible – the salience of various aspects of similarity can be adjusted by changing the weighting of various components in the probe. This is true for both ordinary and contextualized HRRs.

This technique will scale well with the number of episodes in long-term memory for the same reason that convolution and superposition memories scale well with the number of items in clean-up memory (Appendices B and D) – a high-dimensional distributed representation allows exponentially many different patterns.

This work does to some extent contradict Gentner and Markman's [1993] implicit claim that performing structural alignment is a prerequisite to judging analogical similarity. No explicit structural alignment is performed when computing the HRR dot-product similarity estimate, yet the estimate is sensitive to the degree of structural alignment.

The HRR dot-product is not without its drawbacks. First, it is an imperfect measure of similarity, and there are examples where the expected values of HRR dot-products will result in rankings different from SME or ARCS. Second, high-dimensional vectors are required to reduce the noise in estimates to acceptable levels. Computing the dot-products of high-dimensional vectors can still require a significant amount of computation (though I suggest ways to reduce this in Section 3.7). Third, I do not expect the scaling with the number of predicates in an episode to be particularly good – the sum of structural-feature matches becomes a less appropriate measure of similarity as the episodes get larger. A possible solution to this problem is to construct a spreading activation network of HRRs in which each episode is represented as a number of chunks.

Chapter 7

Discussion

In this chapter I briefly discuss several issues which I have not had the opportunity to deal with elsewhere in the thesis: how HRRs can be transformed without decomposition; conflicts between HRRs and some psychological notions of chunks; how other vector-space multiplication operations could be used instead of convolution; how a disordered variant of convolution might be implemented in neural tissue; and weaknesses of HRRs.

7.1 Transformations without decomposition

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

It is easy to do transformations like these with HRRs. Consider Niklasson and van Gelder's task, which was to perform the following transformations:

$$\begin{aligned}
p \to q &\Rightarrow (\neg p \vee q)\\
(\neg p \vee q) &\Rightarrow (p \to q)\\
(p \to (q \vee r)) &\Rightarrow (\neg p \vee (q \vee r))\\
(\neg p \vee (q \vee r)) &\Rightarrow (p \to (q \vee r)\\
(p \to (q \to r) &\Rightarrow (\neg p \vee (q \to r))
\end{aligned}$$

$$(\neg p \vee (q \to r)) \Rightarrow (p \to (q \to r)$$

We need two relations to represent these formulas: implication and disjunction. Implication has two roles, antecedent and consequent, and disjunction also has two roles, negative (negated) and positive (these can be duplicated). I use the following HRRs to represent $(\neg p \vee q)$ and $(p \to (q \vee r))$:

$$\begin{aligned}
\mathbf{R}_{\neg p \vee q} &= \langle \mathbf{disj} + \mathbf{neg} \circledast \mathbf{p} + \mathbf{pos} \circledast \mathbf{q} \rangle\\
\mathbf{R}_{p \to (q \vee r)} &= \langle \mathbf{impl} + \mathbf{ante} \circledast \mathbf{p} + \mathbf{cnsq} \circledast \langle \mathbf{disj} + \mathbf{pos} \circledast \mathbf{q} + \mathbf{pos} \circledast \mathbf{r} \rangle \rangle
\end{aligned}$$

To transform an implication into a disjunction, we need to do three things: change **impl** to **disj**, change **ante** $\circledast$ **x** to **neg** $\circledast$ **x**, and change **cnsq** $\circledast$ **y** to **pos** $\circledast$ **y**. The first can be accomplished by convolving the implication with $\mathbf{impl}^* \circledast \mathbf{disj}$, the second by convolving with $\mathbf{ante}^* \circledast \mathbf{neg}$, and the third by convolving with $\mathbf{cnsq}^* \circledast \mathbf{pos}$. In all cases, there are other convolution products such as $\mathbf{impl}^* \circledast \mathbf{disj} \circledast \mathbf{ante} \circledast \mathbf{x}$, but these products can be treated as noise. These three vectors can be superimposed to give a vector which transforms implications to disjunctions:

$$\mathbf{t}_1 = \langle \mathbf{impl}^* \circledast \mathbf{disj} + \mathbf{ante}^* \circledast \mathbf{neg} + \mathbf{cnsq}^* \circledast \mathbf{pos} \rangle.$$

When we convolve $\mathbf{R}_{\neg p \vee q}$ with $\mathbf{t}_1$, we get a noisy version of $\mathbf{R}_{p \to q}$. A similar vector can be constructed to transform disjunctions to implications:

$$\mathbf{t}_2 = \langle \mathbf{disj}^* \circledast \mathbf{impl} + \mathbf{neg}^* \circledast \mathbf{ante} + \mathbf{pos}^* \circledast \mathbf{cnsq} \rangle.$$

The transformation vectors $\mathbf{t}_1$ and $\mathbf{t}_2$ can be superimposed to give one vector which will transform implications to disjunctions and disjunctions to implications:

$$\mathbf{t} = \langle \mathbf{t}_1 + \mathbf{t}_2 \rangle.$$

This results in more noise products, but these will not make the result unrecognizable if the vector dimension is high enough. The strength of non-noise components in the transformed HRR is $1/\sqrt{k}$ times their strength in the original HRR, where k is the number of components in the transformation vector ($k = 6$ for $\mathbf{t}$).

I simulated the above task, checking the result by decoding its various roles. For example, $\mathbf{t} \circledast \mathbf{R}_{p \to (q \to r)}$ should give $\mathbf{R}_{\neg p \vee (q \to r)}$, which should decode as follows:

$$\begin{aligned}
\mathbf{t} \circledast \mathbf{R}_{p \to (q \vee r)} &\approx \mathbf{disj}\\
\mathbf{t} \circledast \mathbf{R}_{p \to (q \vee r)} \circledast \mathbf{neg}^* &\approx \mathbf{p}\\
\mathbf{t} \circledast \mathbf{R}_{p \to (q \vee r)} \circledast \mathbf{pos}^* &\approx \mathbf{impl}\\
\mathbf{t} \circledast \mathbf{R}_{p \to (q \vee r)} \circledast \mathbf{pos}^* \circledast \mathbf{ante}^* &\approx \mathbf{q}\\
\mathbf{t} \circledast \mathbf{R}_{p \to (q \vee r)} \circledast \mathbf{pos}^* \circledast \mathbf{cnsq}^* &\approx \mathbf{r}
\end{aligned}$$

I tried the various formulae with all possible instantiations of five different variables, which gave 550 different formulae, and 4800 retrieval tests. For 10 runs with $n = 4096$, there were an average of 1.2 retrieval errors per run (out of 4800). Five of the runs had no errors. With lower dimensions, there were more errors. For example, with $n = 2048$ there were an average of 58.5 retrieval errors per run (over 10 runs).

It turns out that the most difficult thing to do is to leave something untransformed. This is because when we leave one thing untransformed, we must leave all the things

superimposed with it untransformed as well. Consider what happens if we want change all **x** to **y**, but leave other things untransformed. Then we must have a transformation vector like this:

$$\mathbf{t} = \mathbf{x}^* \circledast \mathbf{y} + \bar{I}.$$

The identity vector is included so that the result will have an untransformed component of the original. However, this cannot target particular bindings in the original – it applies generally. Thus, when we apply it to a HRR like

$$\mathbf{R} = \mathbf{x} \circledast \mathbf{a} + \mathbf{z} \circledast \mathbf{b}$$

we get

$$\mathbf{t} \circledast \mathbf{R} = \mathbf{y} \circledast \mathbf{a} + \mathbf{x} \circledast \mathbf{a} + \mathbf{z} \circledast \mathbf{b} + \mathbf{noise}.$$

This problem does not arise when we transform everything, because in that case, the cross terms are not similar to anything else we might be using. The only way to solve the problem is to do transformations separately, and clean up intermediate results. It would be interesting to see whether a similar difficulty arises with representations developed by RAAMs.

One thing to note when considering transformations on structures is that result of a transformation will be noisy, and usually can only be cleaned up by decoding and reassembly. This is because the transformed structure will most likely be novel and thus cannot have been stored in clean-up memory. This is different from the situation where we decode chunked structures, and can clean up intermediate structures because they are stored in the clean-up memory.

7.2 Chunks and the organization of long-term memory

7.2.1 The organization of long-term memory

In this thesis, I have been concerned mainly with the internal organization of memory chunks, rather than with how long-term memory for a large set of chunks could be implemented. All I have assumed about long-term (clean-up) memory is that it can keep chunks distinct, and perform closest-match associative retrieval.

I only use superposition and convolution on the small scale, as ways of building chunks with a high degree of internal structure. It seems that we need a third auto-associative operation for long-term memory, if chunks are to be kept distinct. Using the same operator for both the internal organization of chunks and the organization of long-term memory would seem to be prone to ambiguity.

Psychological matrix-based memory models, e.g., those of Halford *et al* [to appear], Humphreys *et al* [1989], and Pike [1984], treat long-term memory as the superposition of chunks (which are the tensor products of their components). This type of scheme appears to have several disadvantages relative to HRRs: chunks have limited and inflexible internal organization; associative retrieval based on partial information is more complicated; and chunks must have very high dimensionality if many of them are to be superimposed and remain distinct. In any case, additional memory mechanisms may be necessary with these models – Halford *et al* recognize the need to clean up the results of their analogy computations. They only need to clean up the representations of atomic objects, but it is hard to see how a principled distinction could be made between atomic and composite objects in anything more complex than a toy system.

7.2.2 The opacity of chunks

Some writers in the psychological literature, e.g., Johnson [1972], have regarded chunks as "opaque containers". This means that a code for chunk reveals nothing about the contents of the chunk until it is unpacked. It implies that the similarity of codes for chunks cannot reflect the similarity of their contents. If we regard chunks as equivalent to reduced descriptions, this is at odds with the desideratum for reduced descriptions that they should give information about their components without decoding, and with the idea that distributed representations should make similarity explicit.

Murdock [1992; 1993] describes the chunks in TODAM2 as opaque, and likens a chunk to objects in a suitcase: to find out what is inside you must open the suitcase. However, TODAM2 chunks are better described as at most semi-opaque, because they do reveal some of their contents without unpacking, and because the similarity of codes for chunks does reflect the similarity of their contents.

From a computational viewpoint, opacity seems to be an undesirable property. Opacity makes retrieval based on partial information about components difficult, and makes transformation without decomposition impossible.

The opacity of chunks has some consequences for the structure of long-term memory: if chunks are opaque, then it is more feasible to superimpose chunks in long-term memory.

7.3 Convolution, tensor products, associative operators, conjunctive coding, and structure

Hinton [1981] and Hinton, McClelland, and Rumelhart [1986] originally described conjunctive coding in terms of outer or tensor products. It is well known that both convolution and the tensor product can be used as the associative operator in pairwise and higher-order associative memories. In this thesis I have tried to show that conjunctive coding can also be based on convolution. It is interesting to consider whether other vector-space multiplication operations could serve as associative operators and as a basis for conjunctive coding. There are two reasons to consider other operations: it widens the space of available associative operators, and it allows us to view some existing network models in these terms.

The properties of convolution which make it a good basis for HRRs are as follows:

1. Bilinearity – $(\alpha\mathbf{a} + \beta\mathbf{b}) \circledast \mathbf{c} = \alpha\mathbf{a} \circledast \mathbf{c} + \beta\mathbf{b} \circledast \mathbf{c}$,[1] for all vectors **a**, **b**, and **c**, and scalars α and β. This makes convolution similarity-preserving – if **a** is similar to $\mathbf{a}'$ (i.e., $\mathbf{a}' = \mathbf{a} + \epsilon\mathbf{b}$ where ϵ is small) then $\mathbf{a} \circledast \mathbf{c}$ will be similar to $\mathbf{a}' \circledast \mathbf{b}$.

2. Invertibility – if vector elements are independently distributed as $N(0, 1/n)$ there is a simple stable approximate inverse of circular convolution.[2] This makes it possible to extract components from HRRs.

3. Randomization – the expected similarity of $\mathbf{a} \circledast \mathbf{b}$ to **a** or **b** is zero (provided that the expected similarity **a** (and **b**) with the identity vector is zero). This helps to avoid unwanted correlations.

[1] Convolution is commutative, so $\mathbf{c} \circledast (\alpha\mathbf{a} + \beta\mathbf{b}) = \alpha\mathbf{c} \circledast \mathbf{a} + \beta\mathbf{c} \circledast \mathbf{a}$ is also true.

[2] An exact inverse also exists, but as explained in Section 3.6.4, it is less stable unless the vectors are constrained to be unitary.

4. Maps onto same space – the circular convolution of two vectors from $\Re^n$ is a vector in $\Re^n$. This allows easy representation of recursive structures.

5. Distribution preservation – if the elements of $\mathbf{a}$ and $\mathbf{b}$ are independently distributed as $N(0, 1/n)$, then the elements of $\mathbf{a} \circledast \mathbf{b}$ will be close to being distributed as $N(0, 1/n)$.[3] The distribution can be made closer by normalizing vectors after convolution. This is equivalent to saying that all the eigenvalues of the operation $\mathbf{f}_a(\mathbf{x}) = \mathbf{a} \circledast \mathbf{x}$ have a magnitude around 1. This property is important for the usability of higher-order associations.

6. Efficient computation – the circular convolution of n-dimensional vectors can be computed in $O(n \log n)$ time using Fast Fourier Transforms.

7. Commutativity and associativity – convolution has both these algebraic properties. While they make the algebra simple, they are not essential properties.

Table 7.1 lists various vector-space multiplication operations, all of which can be viewed as compressions of a tensor product. I would expect that most of the results in this thesis could be achieved with any of the operators from this table taking the place of convolution. The implementation for those which expand vector dimensionality is not as elegant, but it is still possible. To do this, it would be necessary to find distributions of vectors (or matrices) which the operation preserved, and for which there were stable inverses.

Range		Domain	Operations
vector×vector	→	matrix	outer product, tensor product
matrix×vector	→	vector	matrix-vector multiplication
matrix×matrix	→	matrix	matrix multiplication
vector×vector	→	vector	convolution, permuted convolution (Section 3.6.7), random convolution (Section 7.4)

Table 7.1: Vector-space multiplication operations.

It is possible to view Hinton's [1981] triple memory as conjunctive code based on matrix-vector multiplication, in which the PROP units are a reduced representation of the relation.

Smolensky [personal communication] observed that Pollack's [1990] RAAMs can be viewed as using association by matrix-vector multiplication. The fillers (input unit activations) are vectors, and the roles (the weights from input to hidden units) are matrices. The input to the hidden units inputs is the superposition of matrix-vector products of roles and fillers. The weights from the hidden to output units must implement the inverse operation, and clean up output vectors as well. The non-linearities on the hidden and output units probably help with the clean-up, but the also complicate this interpretation.

Recurrent networks can be viewed in a similar fashion, with the recurrent weights implementing a type of trajectory-association (Chapter 5), though again the non-linearities complicate this interpretation.

[3] Distributions are exactly preserved for unitary vectors.

7.4 Implementation in neural tissue

Researchers experimenting with distributed associative memories often succumb to the temptation to explain how their favorite associative memory scheme might be implemented in "neural tissue". I have no greater ability to resist than others.

Recall that the circular convolution of $\mathbf{x}$ and $\mathbf{y}$ can be viewed as a compression of the outer product (Section 3.1). For $n = 4$, the circular convolution of $\mathbf{x}$ and $\mathbf{y}$ is:

$$\begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} x_0y_0 + x_1y_3 + x_2y_2 + x_3y_1 \\ x_0y_1 + x_1y_0 + x_2y_3 + x_3y_2 \\ x_0y_2 + x_1y_1 + x_2y_0 + x_3y_3 \\ x_0y_3 + x_1y_2 + x_2y_1 + x_3y_0 \end{bmatrix}$$

Each z_i is the sum of n x_jy_k terms, and each x_j (and y_k) appears exactly once in the sum. Each product appears in the sum for only one z_i. Decoding by correlation works (when the x_i and y_i are independently distributed as $N(0, 1/n)$) because $\mathbf{x} \circledast \mathbf{z} \approx \mathbf{y}$, since

$$\mathbf{x} \circledast \mathbf{z} = \begin{bmatrix} (x_0^2 + x_1^2 + x_2^2 + x_3^2)y_0 + \ldots \\ (x_0^2 + x_1^2 + x_2^2 + x_3^2)y_1 + \ldots \\ (x_0^2 + x_1^2 + x_2^2 + x_3^2)y_2 + \ldots \\ (x_0^2 + x_1^2 + x_2^2 + x_3^2)y_3 + \ldots \end{bmatrix} = (1 + \zeta)\mathbf{y} + \bar{\eta}$$

where ζ and the η_i can be regarded as zero-mean Gaussian noise (Section 3.1.2).

The specific regular nature of the outer product compression makes it possible to perform convolution quickly via FFTs. It seems unlikely that either FFTs or this highly-ordered compression could be happening anywhere in the brain.

It turns out that it is not necessary to compress the outer product in such a highly ordered manner. Even if the terms in the sum for z_i are randomly selected, the resulting $\mathbf{z}$ vector is an association of $\mathbf{x}$ and $\mathbf{y}$ which has all the essential properties of convolution. The decoding noise is the same as for circular convolution if (a) each x_j (and y_k) appears exactly once in the sum for each z_i, and (b) each product appears in the sum for only one z_i. However, these conditions do not appear to be essential – breaking them will just result in more noise.

Consider the disordered outer product compression $\mathbf{z} = \mathbf{x} \circledast_d \mathbf{y}$ in Figure 7.1. The expressions for the z_i are as follows:

$$\begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} x_0y_3 + x_1y_2 + x_2y_0 + x_3y_1 \\ x_0y_2 + x_1y_1 + x_2y_3 + x_3y_0 \\ x_0y_0 + x_1y_3 + x_2y_1 + x_3y_2 \\ x_0y_1 + x_1y_0 + x_2y_2 + x_3y_3 \end{bmatrix}$$

To reconstruct $\mathbf{y}$ from $\mathbf{z}$ (= $\mathbf{x} \circledast_d \mathbf{y}$) and $\mathbf{x}$, we need to find the appropriate compression, $\oplus_d$, of the outer product of $\mathbf{z}$ and $\mathbf{x}$ (i.e., the analogue of correlation), so that $\mathbf{x} \oplus_d \mathbf{z} \approx \mathbf{y}$. Call the elements of this vector $[\mathbf{x} \oplus_d \mathbf{z}]_i$. Then we want

$$[\mathbf{x} \oplus_d \mathbf{z}]_i = (x_0^2 + x_1^2 + x_2^2 + x_3^2)y_i + \ldots$$

where the dots are $x_jx_ky_l$ terms which can be treated as zero-mean Gaussian noise.

We can find this compression by selecting for $[\mathbf{x} \oplus_d \mathbf{z}]_i$ those terms in the outer product of $\mathbf{x}$ and $\mathbf{z}$ which contain $x_j^2y_i$. For example, we want $[\mathbf{x} \oplus_d \mathbf{z}]_0 = x_2z_0 + x_3z_1 + x_0z_2 + x_1z_3$. The

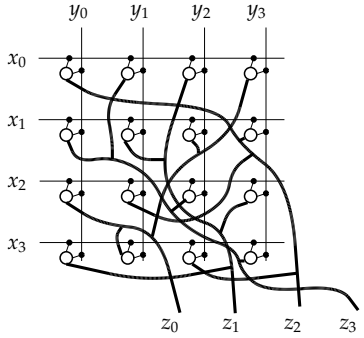


Figure 7.1: A disordered compression of the outer product of **x** and **y**.

x_2z_0 term is present because x_2y_0 appears in z_0, the x_3z_1 term because x_3y_0 appears in z_1, the x_0z_2 term because x_0y_0 appears in z_2, and the x_1z_3 term because x_1y_0 appears in z_3. The whole expression for $\mathbf{x} \circledast_d \mathbf{z}$ is:

$$\mathbf{x} \circledast_d \mathbf{z} = \begin{bmatrix} x_2z_0 + x_3z_1 + x_0z_2 + x_1z_3 \\ x_3z_0 + x_1z_1 + x_2z_2 + x_0z_3 \\ x_1z_0 + x_0z_1 + x_3z_2 + x_2z_3 \\ x_0z_0 + x_2z_1 + x_1z_2 + x_3z_3 \end{bmatrix}$$

This expands to

$$\mathbf{x} \circledast_d \mathbf{z} = \begin{bmatrix} (x_0^2 + x_1^2 + x_2^2 + x_3^2)y_0 + \dots \\ (x_0^2 + x_1^2 + x_2^2 + x_3^2)y_1 + \dots \\ (x_0^2 + x_1^2 + x_2^2 + x_3^2)y_2 + \dots \\ (x_0^2 + x_1^2 + x_2^2 + x_3^2)y_3 + \dots \end{bmatrix} = (1 + \zeta)\mathbf{y} + \bar{\eta}$$

where ζ and $\bar{\eta}_i$ can be treated as zero mean noise, assuming the x_i and y_i are independently distributed as $N(0, 1/n)$.

Turning to the matter of neural implementation, the sum-of-products could be computed by *sigma-pi* units. Feldman and Ballard [1982] proposed this type of unit and suggested the sum-of-product interactions might happen in the dendritic tree. In neural tissue, the encoding map could be fixed. The appropriate decoding map could be learned, provided one started with sigma-pi units which had a dense sampling of z_jx_k products. The learning would attenuate the contribution of inappropriate z_jx_k terms in $[\mathbf{x} \circledast_d \mathbf{z}]_i$. This could be done by simple Hebbian learning in an auto-associative framework – it would not require backpropagation of errors through multiple layers. A considerable amount of training would be required, since there would be n^3 parameters. However, auto-associations of random vectors would suffice for training examples, so there should be no problem getting enough training data.

7.5 Weaknesses of HRRs

HRRs have several weaknesses as a representation for nested relations. While these are not fatal, it is good to be aware of them.

- Representing types and tokens as a superposition of features (Section 3.5) makes superpositions of tokens subject to crosstalk. To a large extent, this problem can be avoided by making sure that objects are bound to different roles. A further possible remedy is to give tokens more internal structure, e.g., by convolving features together.
- Relations which have identical or symmetric roles (e.g., conjunction, same-as, addition) have identical role vectors. In a HRR, this has the same effect and problems as superimposing the fillers of these roles.
- The decoding of binding chains for nested structures can be ambiguous, because of the commutativity and associativity of convolution. This is mainly a problem when one relation appears more than once in a HRR – e.g., the agent of the object has the same binding chain as the object of the agent (since $\mathbf{r}_1 \circledast \mathbf{r}_2 \circledast \mathbf{x} = \mathbf{r}_2 \circledast \mathbf{r}_1 \circledast \mathbf{x}$). One solution to this problem is to use chunking. Another is to use a non-commutative version of convolution (Section 3.6.7).
- If there can be more than one object retrieved from clean-up memory (as happens when there are identical role vectors and the filler is decoded), we need to set cutoff threshold below which similarity is judged to be merely due to noise. It is possible to analyze a particular scenario to derive an appropriate threshold, but this is probably impractical in general.

In the literature, there are several claims about weaknesses of conjunctive codes and HRRs which I believe are not true. Hummel and Holyoak [1992] claim that the most serious limitation of conjunctive codes is that they lose the natural similarity structure of predicates and objects. By this, they mean that the proposition `(chase Arnold Bill)` is naturally more similar to `(chase Bill Arnold)` than to `(says My-doctor Caffeine-makes-me-nervous)`. While this claim might be true for some implementations of conjunctive coding, I showed in Chapter 6 how the natural similarity structure of predicates and objects can be neatly captured by conjunctive coding in the form of HRRs. On the topic of how this is achieved, there is a tension between wanting the bindings of a filler with different roles (e.g., **bill** bound to **agent** versus **bill** bound to **object**) to be similar (to aid with the recognition of similarity) and wanting them to be different (to be able to tell which role the filler is bound to). In HRRs, different components are responsible for each of these (**bill** for the similarity to other predicates involving Bill, and **bill** $\circledast$ **agent** for the role binding), and they can be weighted appropriately.

Smolensky, Legendre and Miyata [1992] remark that HRRs appeared to be an adequate representation, "at least for stimuli with only a small degree of structure" (p21-22). HRRs can in fact cope with quite a high degree of structure, both in depth and width. Without chunking, the width of structures is limited by the vector dimension (the relationship is linear). HRRs cope quite well with deep structures if vectors are either normalized or constrained to be unitary. With chunking, structures of unlimited depth and width can be stored, by breaking them up into chunks of manageable size.

7.6 Conclusion

HRRs provide a solution to the longstanding problem of how to encode nested relational structure (i.e., compositional structure) in a fixed-width distributed representation. HRRs can be seen as an implementation of Hinton's reduced representations, and satisfy all the

desiderata for reduced representations: adequacy, reduction, systematicity, and informativeness. HRRs inherit all the major advantages of distributed representations of atomic objects: they allow explicit representation of relevant aspects, they make similarity among composite objects explicit, they store information in a redundant fashion, they use representational resources efficiently, and they exist in a continuous vector space. HRRs are based on circular convolution, a bilinear associative memory operator. It is possible to analyze from the first principles the properties of systems which use HRRs. The storage capacity and probabilities of correct operation can be derived from the analysis.

One of the major drawbacks of convolution-based memory is the low signal-to-noise ratio of the results of decoding. However this problem can be overcome by using HRRs in conjunction with an auto-associative memory to clean up decoding results. Storing HRRs and atomic objects[4] in the auto-associative memory allows for a complex network of relationships among the entities in memory.

HRRs encode nested structures in such a way that many aspects of the underlying structure are made explicit in the surface form of the representation. This allows the degree of structural alignment (i.e., structural similarity) between two HRRs to be estimated by their dot-product. Thus, the use of HRRs allows the results of a exponential-time computation (structure matching) to be estimated by a linear-time computation. This technique has possible applications as a filter in psychological models of memory retrieval, and in reasoning systems.

HRRs make it easy to encode structure in a distributed representation. Although I have not investigated learning over hierarchical structures, HRRs have the potential to allow a learning system to devote its computational resources to the truly difficult task of learning about the relationships in the stimuli it is exposed to, rather than merely learning how to represent the stimuli.

[4] Whether or not atomic, or base-level, objects are useful in sophisticated systems, or have any psychological reality, does not really matter in this scheme.

Appendix A

Means and variances of similarities between bindings

The means and variances of the dot-products of convolution bindings can be calculated using the expressions in Table 3.1. For the calculation of the variance of the dot-product of $\mathbf{B}_1$ and $\mathbf{B}_4$ from Section 3.10.5 we need the following two expressions from Table 3.1:

$$\begin{aligned} \mathrm{var}[(\mathbf{a} \circledast \mathbf{b}) \cdot (\mathbf{a} \circledast \mathbf{b})] &= \frac{6n+4}{n^2} \\ \mathrm{var}[(\mathbf{a} \circledast \mathbf{b}) \cdot (\mathbf{a} \circledast \mathbf{c})] &= \frac{2n+2}{n^2} \end{aligned}$$

The variance of $\mathbf{B}_1 \cdot \mathbf{B}_4$ is:

$$\begin{aligned} &\mathrm{var}[(\mathbf{eat}_{agt} \circledast \mathbf{mark}) \cdot (\mathbf{eat}_{agt} \circledast \mathbf{john})] \\ &= \mathrm{var}[(\mathbf{eat}_{agt} \circledast (\mathbf{person} + \mathbf{id}_{mark})/\sqrt{3}) \cdot (\mathbf{eat}_{agt} \circledast (\mathbf{person} + \mathbf{id}_{john})/\sqrt{3})] \\ &= \mathrm{var}[2/3\mathbf{eat}_{agt} \circledast \mathbf{person} \cdot \mathbf{eat}_{agt} \circledast \mathbf{person}] \\ &\quad + \mathrm{var}[\sqrt{2}/3\mathbf{eat}_{agt} \circledast \mathbf{person} \cdot \mathbf{eat}_{agt} \circledast \mathbf{id}_{john}] \\ &\quad + \mathrm{var}[\sqrt{2}/3\mathbf{eat}_{agt} \circledast \mathbf{id}_{mark} \cdot \mathbf{eat}_{agt} \circledast \mathbf{person}] \\ &\quad + \mathrm{var}[1/3\mathbf{eat}_{agt} \circledast \mathbf{id}_{mark} \cdot \mathbf{eat}_{agt} \circledast \mathbf{id}_{john}] \\ &= \frac{4}{9}\frac{6n+4}{n^2} + \frac{2}{9}\frac{2n+2}{n^2} + \frac{2}{9}\frac{2n+2}{n^2} + \frac{1}{9}\frac{2n+2}{n^2} = \frac{34n+26}{9n^2} \end{aligned}$$

The variances of the dot-products of $\mathbf{B}_1$ with other bindings from Section 3.10.5 can be calculated in a similar fashion and are shown in Table A.1. The sample means and variances from 10,000 trials are shown in the same table. The sample statistics agree with the derived expressions within the margin of error. (The standard deviation of a variance statistic for samples from a normal distribution is approximately $\sqrt{2/N}\sigma^2$, where N is the number of samples. This works out to about 1.4% of the variances here.)

Dot-product ($\mathbf{x}$ = **mark** $\circledast$ **eat**$_{agt}$)	Expected value	Variance	Variance (n = 512)	Sample mean	Sample variance
$\mathbf{x} \cdot$ **mark** $\circledast$ **eat**$_{agt}$	1	$(6n+4)/(n^2)$	0.0117	1.0009	0.0118
$\mathbf{x} \cdot$ **mark** $\circledast$ **see**$_{agt}$	1/2	$(6xn+5)/(2n^2)$	0.00586	0.5003	0.00586
$\mathbf{x} \cdot$ **mark** $\circledast$ **eat**$_{obj}$	0	$(2n+2)/(n^2)$	0.00391	0.0010	0.00385
$\mathbf{x} \cdot$ **john** $\circledast$ **eat**$_{agt}$	2/3	$(34n+26)/(9n^2)$	0.00739	0.6665	0.00749
$\mathbf{x} \cdot$ **john** $\circledast$ **see**$_{agt}$	1/3	$(73n+50)/(36n^2)$	0.00397	0.3328	0.00398
$\mathbf{x} \cdot$ **john** $\circledast$ **eat**$_{obj}$	0	$(13n+8)/(9n^2)$	0.00283	0.0003	0.00281
$\mathbf{x} \cdot$ **the_fish** $\circledast$ **eat**$_{agt}$	0	$(2n+2)/(n^2)$	0.00391	0.0009	0.00398
$\mathbf{x} \cdot$ **the_fish** $\circledast$ **see**$_{agt}$	0	$(5n+2)/(4n^2)$	0.00244	0.0004	0.00253
$\mathbf{x} \cdot$ **the_fish** $\circledast$ **eat**$_{obj}$	0	$n/(n^2)$	0.00195	0.0000	0.00192

Table A.1: Means and variances of the dot-products of the bindings. The sample statistics are from a 10,000 runs (i.e., different choices of base vectors) with 512-dimensional vectors.

Appendix B

The capacity of a superposition memory

In Section 3.2.1 I discussed how the probability of correct recognition in a superposition memory could be calculated. Knowing how to calculate this probability enables us to calculate the capacity, i.e., how many vectors can be stored, for a given dimension and probability of error. However, the formula for this probability involves an optimization, and it is difficult to find an analytic form for the solution. In the first section of this Appendix I report the results of numerical solutions of this formula. These results show how the number of vectors stored in the trace, and the number of vectors in the clean-up memory, scale with vector dimension. I call this a *simple* superposition memory because I assume that all the vectors in the clean-up memory are independently chosen, i.e., there is no systematic similarity among them. In Appendix C, I gave an analytic lower bound on the capacity of a simple superposition memory.

In the second section of this Appendix I calculate the recognition probabilities and scaling properties for superposition memory in the case where there is similarity among the vectors in the clean-up memory.

B.1 Scaling properties of a simple superposition memory

I use the following symbols and definitions (which are the same as in Section 3.2.1):

- n, the vector dimension. "Random" vector have elements independently drawn from $N(0, 1/n)$.
- $\mathbb{E}$, a set of m random vectors, $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$ etc. These are the vectors stored in clean-up memory.
- $\mathbf{t}$, a memory trace which is the (unnormalized) superposition of k distinct vectors from $\mathbb{E}$.
- Pr(All Correct), the probability of correctly determining which vectors are and are not stored in the memory trace.

Equation 3.2 in Section 3.2.1 gives the probability of correct recognition in a simple superposition memory:

$$\text{Pr(All Correct)} \quad = \quad \text{Pr(Hit)}^k \, \text{Pr(Reject)}^{m-k}$$

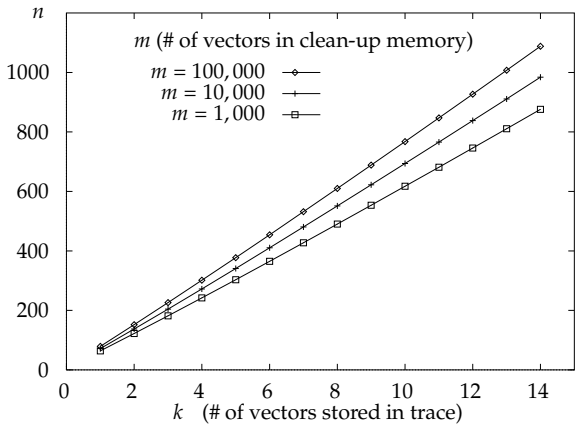


Figure B.1: Scaling of n with k for Pr(Error) = 0.01 with various m in a simple superposition memory. The threshold was adjusted to maximize the capacity for each data point.

$$= \max_t \Pr(s_a > t)^k \Pr(s_r < t)^{m-k}$$

I wrote a computer program to find the threshold (t) which maximizes Pr(All Correct) for given values of k, m, and n. I used it to find the vector dimension (n) which gives Pr(All Correct) = 0.99 for various values of k and m. The results are plotted in Figures B.1 and B.2. The threshold was adjusted to maximize the capacity for each data point.

The scaling of k with n is slightly less than linear (Figure B.1). This scaling is due to two factors. The linear part is due to the variances being equal to the ratio of k to n (in the limit). This means that if k is increased by some factor, then n should be increased by the same factor to keep the variance constant. The sublinear part is due to the exponentiation of Pr(Hit) by k in the equation for Pr(All Correct). When the threshold is chosen to optimize Pr(All Correct), Pr(Hit) is much smaller than 1 − Pr(False Alarm). If k increases, while the ratio k/n is held constant, Pr(All Correct) decreases.

The scaling of m with respect to n is very good. m can increase exponentially with n while maintaining Pr(All Correct) constant. This scaling is shown in Figure B.2. The exponential nature of the scaling is due to the rapid drop-off of the area under the tail of the normal distribution.

The scaling of Pr(Error) with n is also very good. The probability of error drops exponentially with linear increase in n, as illustrated in Figure B.3.

B.1.1 Computation involved

Computing the dot-product of the trace with all the vectors in $\mathbb{E}$ may seem like an unreasonable amount of computation. However, any process which does closest match retrieval must effectively do this. If we have a particularly efficient closest match process we can

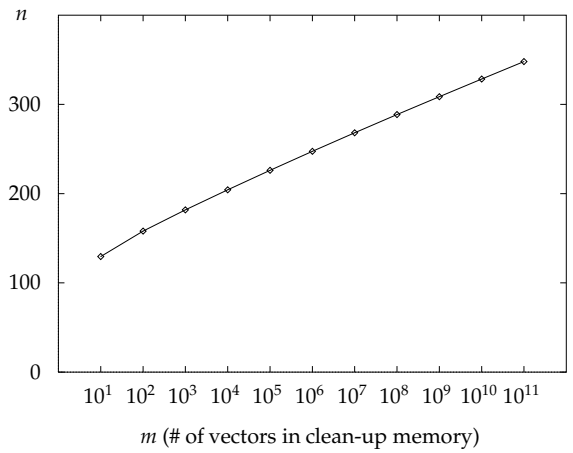


Figure B.2: Scaling of n with m for k = 3 and Pr(Error) = 0.01 in a simple superposition memory.

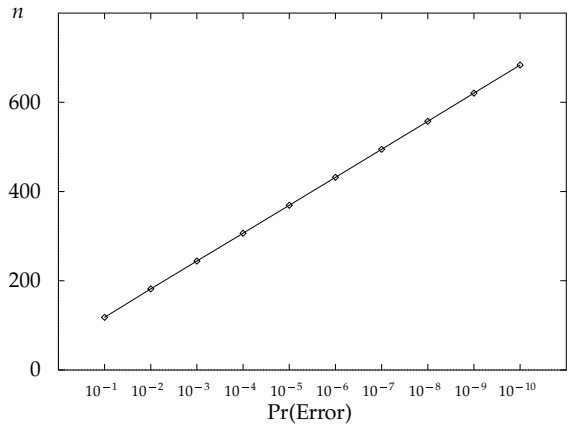


Figure B.3: Scaling of n with Pr(Error) for k = 3 and m = 1000 in a simple superposition memory.

in most situations use it to quickly find all the vectors whose dot-product with the trace exceeds some threshold. Furthermore, this is a computation which is very amenable to implementation in parallel hardware.

B.2 A superposition memory with similarity among the vectors

When there is similarity among vectors the analysis becomes more complex, because recognition signals will come from more than two distributions. Having similarity among vectors is the rule rather than the exception, so it is important to know how similarity affects the scaling properties.

Suppose than in our set $\mathbb{E}$ of m vectors there is a subset $\mathbb{E}_p$ of m_p vectors which are similar to each other. Suppose the vectors in $\mathbb{E}_p$ have the general form $\mathbf{p}_i = \alpha\mathbf{p} + \beta\mathbf{d}_i$, where $\mathbf{p}$ is the component the vectors in $\mathbb{E}_p$ have in common, $\mathbf{d}_i$ is a random vector, and the scalars α and β are positive weighting constants such that $\alpha^2 + \beta^2 = 1$ (so that $\mathrm{E}[|\mathbf{p}_i|] = 1$). The expected value of the dot-product of $\mathbf{p}_i$ and $\mathbf{p}_j \in \mathbb{E}_p$ is α^2. The other $m - m_p$ vectors in $\mathbb{E}$ are randomly chosen. The expected value of the dot-product of two different vectors from $\mathbb{E}$ is zero, provided at least one them is not from $\mathbb{E}_p$.

Since the vectors in $\mathbb{E}_p$ are similar, they are much more likely to be confused with each other than with random vectors. This results in a lower capacity for a superposition memory storing similar vectors than for a superposition memory storing random vectors.

In this analysis I consider only the situation in which all the vectors stored in the trace are from $\mathbb{E}_p$. The reasons for this are related to the limitations of superposition memories and will be explained later.

Suppose we have stored k distinct vectors from $\mathbb{E}_p$ in the trace, so that

$$\mathbf{t} = \sum_{i=1}^{k} \mathbf{p}_i \quad \mathbf{p}_i \in \mathbb{E}_p.$$

The signal for probing the trace with a vector $\mathbf{x}$ will be distributed in one of three ways, depending on whether $\mathbf{x}$ is one of the vectors stored in the trace, or is similar to the stored vectors (i.e., $\mathbf{x}$ is some other element of $\mathbb{E}_p$,) or is some other non-similar element of $\mathbb{E}$.

Since $\mathbf{p}$ can be a component of both the trace and the probe, the value $\mathbf{p} \cdot \mathbf{p}$ appears in both the accept and the reject-similar signals. This value will be a constant for constant $\mathbf{p}$. Better discrimination can be achieved if this value is taken into account when the threshold is chosen. One way of doing this is to choose $\mathbf{p}$ so that $\mathbf{p} \cdot \mathbf{p}$ is exactly equal to 1, which is what I assume in this analysis.

Let s_a (the accept signal) be the signal for probing with a vector stored in the trace. Without loss of generality, we can assume that $\mathbf{p}_1$ is in the trace and that the probe is equal to $v\mathbf{p}_1$. Assuming that $\mathbf{p} \cdot \mathbf{p} = 1$ (and thus $\mathrm{var}[\mathbf{p} \cdot \mathbf{p}] = 0$), the mean and variance of s_a are:

$$\begin{aligned}
\mathrm{E}[s_a] &= \mathrm{E}[\mathbf{t} \cdot \mathbf{p}_1] \\
&= \mathrm{E}[\sum_{i=1}^{k} \mathbf{p}_i \cdot \mathbf{p}_1] \\
&= \mathrm{E}[(k\alpha\mathbf{p} + \sum_{i=1}^{k} \mathbf{d}_i) \cdot (\alpha\mathbf{p} + \beta\mathbf{d}_1)] \\
&= k\alpha^2\mathrm{E}[\mathbf{p} \cdot \mathbf{p}] + k\alpha\beta\mathrm{E}[\mathbf{p} \cdot \mathbf{d}_1] + \alpha\beta\sum_{i=1}^{k}\mathrm{E}[\mathbf{d}_i \cdot \mathbf{p}] + \beta^2\mathrm{E}[\mathbf{d}_1 \cdot \mathbf{d}_1] + \beta^2\sum_{i=2}^{k}\mathrm{E}[\mathbf{d}_i \cdot \mathbf{d}_1] \\
&= k\alpha^2 + 0 + 0 + \beta^2 + 0 \\
&= k\alpha^2 + \beta^2 \\
\mathrm{var}[s_a] &= \mathrm{var}[\mathbf{t} \cdot \mathbf{p}_a] \\
&= \mathrm{var}[(k\alpha\mathbf{p} + \sum_{i=1}^{k} \mathbf{d}_i) \cdot (\alpha\mathbf{p} + \beta\mathbf{d}_1)] \\
&= k^2\alpha^4\mathrm{var}[\mathbf{p} \cdot \mathbf{p}] + (k+1)^2\alpha^2\beta^2\mathrm{var}[\mathbf{p} \cdot \mathbf{d}_1] \\
&+ \quad + \alpha^2\beta^2\sum_{i=2}^{k}\mathrm{var}[\mathbf{d}_i \cdot \mathbf{p}] + \beta^4\mathrm{var}[\mathbf{d}_1 \cdot \mathbf{d}_1] + \beta^4\sum_{i=2}^{k}\mathrm{var}[\mathbf{d}_i \cdot \mathbf{d}_1] \\
&= \frac{1}{n}(0 + (k+1)^2\alpha^2\beta^2 + (k-1)\alpha^2\beta^2 + 2\beta^4 + (k-1)\beta^4 \\
&= \frac{1}{n}((k^2+3k)\alpha^2\beta^2 + (k+1)\beta^4)
\end{aligned}$$

Signal	E	var
s_a	$k\alpha^2 + \beta^2$	$\frac{1}{n}(k(k+3)\alpha^2\beta^2 + (k+1)\beta^4)$
r_s	$k\alpha^2$	$\frac{1}{n}(k(k+1)\alpha^2\beta^2 + k\beta^4)$
r_d	0	$\frac{1}{n}(k^2\alpha^2 + k\beta^2)$

Table B.1: Means and variances of signals for a superposition memory with similarity.

There are two other signals; the reject-similar signal r_s, for when the probe is from $\mathbb{E}_p$ but is not in the trace, and the reject-dissimilar signal r_d, for when the probe is not from $\mathbb{E}_p$. The means and variances of the three signals are listed in Table B.1. The probability of correctly identifying all the vectors in the trace, for a threshold t is

$$\Pr(\text{All Correct}) = \Pr(s_a > t)^k \Pr(r_s < t)^{m_p - k} \Pr(r_d < t)^{m - m_p}.$$

The distributions of s_a, r_s, and r_d, for $n = 512$, $m = 1000$, $m_p = 100$, $\alpha^2 = 0.5$, and $k = 3$, are shown in Figure B.4. For these parameter values, the optimal threshold is 1.79 and the probability of correctly identifying all the components of $\mathbf{t}$ is 0.906.

The way n must vary with α to give a constant Pr(All Correct) is shown in Figure B.5. As α gets larger the similarity between the elements of $\mathbb{E}_p$ grows, and it is necessary to use much higher dimensional vectors to discriminate them.

The scaling of n with k is shown in Figure B.6. The required dimensionality of the vectors is proportional to k^2. This is because the variances are proportional to the ratio of k^2 to n, which in turn is due to the component $k\alpha\mathbf{p}$ in the trace.

B.2.1 The effect of the decision process on scaling

It is possible to design more complex decision procedures that increase the probability of making correct decisions. This was done to some extent in the above analysis by assuming

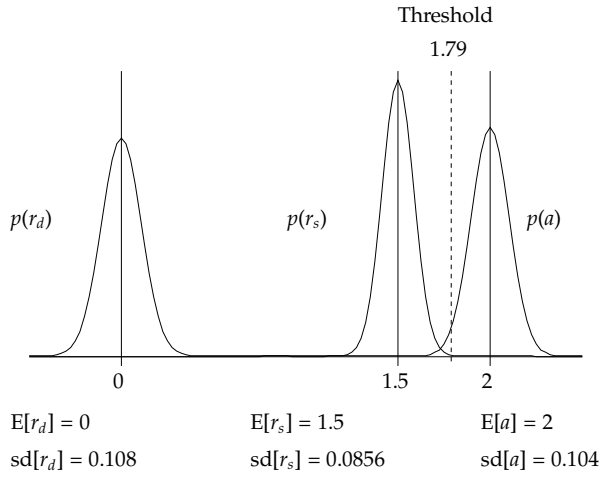


Figure B.4: Probability density functions for signals in an superposition memory with similarity among vectors, with n = 512, m = 1000, $|\mathbb{E}_p|$ = 100, α^2 = 0.5, and k = 3.

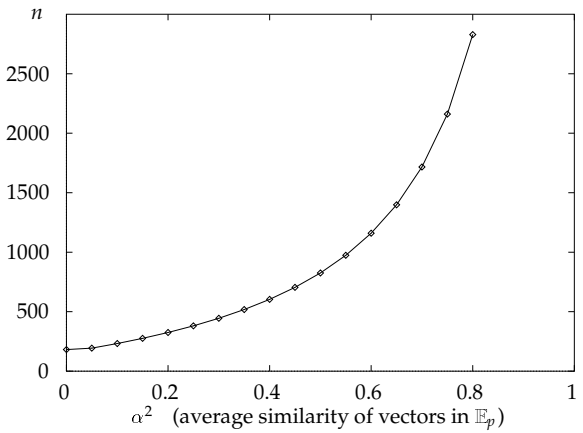


Figure B.5: Scaling of n with α for k = 3, m = 1000, and Pr(Error) = 0.01 in a superposition memory with similarity.

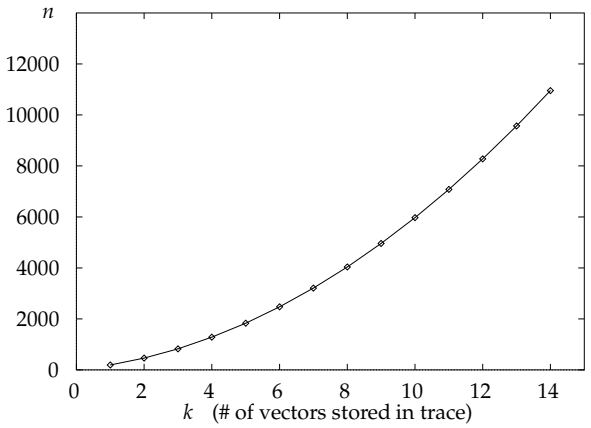


Figure B.6: Scaling of n with k for α^2 = 0.5, m = 1000, and Pr(Error) = 0.01 in a superposition memory with similarity.

that $\mathbf{p} \cdot \mathbf{p} = 1$. This has nearly the same effect as taking into account the value of $\mathbf{p} \cdot \mathbf{p}$ when setting the threshold.

Another way to improve the decision procedure would be to take into account the value of $(\mathbf{a}+\mathbf{b}+\mathbf{c}) \cdot \mathbf{p}$ when setting the threshold. For a particular trace, s_a and r_s have the common term $\alpha\beta(\mathbf{a} + \mathbf{b} + \mathbf{c}) \cdot \mathbf{p}$. This will cause the values of s_a and r_s to be correlated, and their distributions will be tighter than shown in Figure B.4 (the distributions in Figure B.4 are for a randomly selected trace.) This will increase the probability of making correct decisions, but the scaling of n with k will remain quadratic.

A way to more radically improve the decision procedure would be to subtract $k\alpha\mathbf{p}$ from the trace, and then probe it with the token parts of the vectors from $\mathbb{E}_p$, i.e., $\mathbf{a}$ rather than $\mathbf{p}_a$. This would not only increase the probability of making correct decisions, but would also make the scaling of n with k near to linear. However, it would be a far more complicated procedure.

Another point worth noting is that a k-closest match procedure (i.e. choosing the k members of $\mathbb{E}$ that have the highest dot-product with $\mathbf{t}$) will give better results than a procedure using threshold tests. For any particular trace it is more likely that all the accept signals are greater than all the reject signals than that all the accept signals are greater than some threshold and all the reject signals are less than some threshold. A problem with this type of procedure is knowing the value of k. Unless there is some other way of knowing how many items to accept, some type of threshold is necessary.

B.3 Limitations of superposition memories

Superposition memories suffer badly from crosstalk when the vectors stored are similar. This is reason that the means of s_a and r_s (in the previous section) are so high. The significance of this limitation becomes clear if we consider what happens when we try to store two vectors from $\mathbb{E}_p$ (whose average similarity $\alpha^2 = 0.5$) and one vector from $\mathbb{E} - \mathbb{E}_p$, e.g., $\mathbf{t} = \mathbf{p}_a + \mathbf{p}_b + \mathbf{c}$. There will be two distributions of accept signals – the accept similar (a_s) and the accept dissimilar signals (a_d). The problem is that the accept dissimilar signal ($a_d = (\mathbf{p}_a + \mathbf{p}_b + \mathbf{c}) \cdot \mathbf{c}$)) will have the same mean (=1.0) as the reject similar signal ($r_s = (\mathbf{p}_a + \mathbf{p}_b + \mathbf{p}_x) \cdot \mathbf{x}, \mathbf{p}_x \in (\mathbb{E}_p - \{\mathbf{p}_a, \mathbf{p}_b\})$). Thus, no single-threshold or k-closest-match procedure will be able to discriminate the signals. A possible solution is to use a high threshold when probing with a similar vector and a lower threshold when probing with a dissimilar vector. However, this makes for a very complex decision process. A better solution is to avoid the situation where vectors of high similarity are superimposed.

Appendix C

A lower bound for the capacity of superposition memories

In this Appendix I derive a lower bound (Inequality C.8) on the number of vectors that can be stored in a simple superposition memory, for given vector dimension, probability of error, and total number of vectors in clean-up memory.

I use the following symbols and definitions (which are the same as in Section 3.2.1 and Appendix B):

- n, the vector dimension. "Random" vector have elements independently drawn from $\mathrm{N}(0, 1/n)$.
- $\mathbb{E}$, a set of m random vectors, $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$ etc. These are the vectors stored in clean-up memory.
- $\mathbf{t}$, a memory trace which is the (unnormalized) superposition of k distinct vectors from $\mathbb{E}$.
- Pr(All Correct), the probability of correctly determining which vectors are and are not stored in the memory trace.
- $q = 1 - \Pr(\text{All Correct})$
- s_a and s_r, the accept and reject signals: $s_a \stackrel{d}{=} \mathrm{N}(1, (k+1)/n)$ and $s_r \stackrel{d}{=} \mathrm{N}(0, k/n)$.
- erfc(x), the standard "error function": $\mathrm{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} \mathrm{d}t$
- tail(x), the area under the (normalized) normal probability density function beyond x (in one tail):

$$\mathrm{tail}(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{\frac{-t^2}{2}} \mathrm{d}t = \frac{1}{2} \mathrm{erfc}\left(\frac{x}{\sqrt{2}}\right)$$

Equation 3.2 in Section 3.2.1 gives the probability of correct recognition in a simple superposition memory:

$$\begin{aligned} \Pr(\text{All Correct}) &= \Pr(\text{Hit})^k \Pr(\text{Reject})^{m-k} \\ &= \max_t \Pr(s_a > t)^k \Pr(s_r < t)^{m-k} \end{aligned}$$

I use the following inequality from Abramowitz and Stegun [1965], and a simplification of it:

$$\begin{aligned} \text{erfc}(x) &< \frac{2}{\sqrt{\pi}} e^{-x^2} \frac{1}{x + \sqrt{x^2 + \frac{4}{\pi}}} \\ \text{erfc}(x) &< \frac{1}{x\sqrt{\pi}} e^{-x^2} \end{aligned} \tag{C.1}$$

The first step in finding a lower bound for Pr(All Correct) is to use a threshold of 0.5. Pr(All Correct) must be greater than the probability of correctly discriminating the signals using a non-optimal threshold. The second step involves several applications of the inequality $(1-\epsilon)^k > 1 - k\epsilon$ (which is true for $0 \le \epsilon \le 1$).

$$\begin{aligned} \Pr(\text{All Correct}) &= \max_t \Pr(s_a > t)^k \Pr(s_r < t)^{m-k} && \text{(C.2)} \\ &> \Pr(s_a > 0.5)^k \Pr(s_r < 0.5)^{m-k} \\ &= (1 - \Pr(s_a < 0.5))^k (1 - \Pr(s_r > 0.5))^{m-k} \\ &> (1 - k\Pr(s_a < 0.5))(1 - (m-k)\Pr(s_r > 0.5)) \\ &> 1 - k\Pr(s_a < 0.5) - (m-k)\Pr(s_r > 0.5) && \text{(C.3)} \end{aligned}$$

Now consider q, the probability of one or more errors. This can be simplified by first using Inequality C.3 to give Inequality C.4, then next replacing smaller variances with the maximum variance to give Inequality C.5. After that we use Inequality C.1 to give Inequality C.6 and finally replace the square root factor by one to give Inequality C.7, since it is safe to assume that that factor is less than 1.

$$\begin{aligned} q &= 1 - \Pr(\text{All Correct}) \\ &< k\Pr(s_a < 0.5) + (m-k)\Pr(s_r > 0.5) && \text{(C.4)} \\ &= k\,\text{tail}\left(\frac{1}{2}\sqrt{\frac{n}{k+1}}\right) + (m-k)\text{tail}\left(\frac{1}{2}\sqrt{\frac{n}{k}}\right) \\ &< k\,\text{tail}\left(\frac{1}{2}\sqrt{\frac{n}{k+1}}\right) + (m-k)\text{tail}\left(\frac{1}{2}\sqrt{\frac{n}{k+1}}\right) && \text{(C.5)} \\ &= m\,\text{tail}\left(\frac{1}{2}\sqrt{\frac{n}{k+1}}\right) \\ &= \frac{m}{2}\text{erfc}\left(\frac{1}{2}\sqrt{\frac{n}{2(k+1)}}\right) \\ &< \frac{m}{\sqrt{\pi}}\sqrt{\frac{2(k+1)}{n}}\; e^{\frac{-n}{8(k+1)}} && \text{(C.6)} \\ &< m\, e^{\frac{-n}{8(k+1)}} \qquad \text{if } \sqrt{\frac{2(k+1)}{\pi n}} < 1 && \text{(C.7)} \end{aligned}$$

Rearranging gives:

$$\begin{aligned} n &< 8(k+1)\ln(\frac{m}{q}) \qquad \text{if } n > \frac{2(k+1)}{\pi} \\ \text{or} \quad k &> \frac{n}{8\ln(m/q)} - 1 \qquad \text{if } k < \frac{n\pi}{2} - 1 && \text{(C.8)} \end{aligned}$$

This lower bound on the capacity (k) is reasonably close. Numerical solutions of the exact expression for Pr(All Correct) (Equation C.2) for k in the range (2..14), m in $(10^2 \ldots 10^{10})$, and q in $(10^{-2} \ldots 10^{-10})$ are reasonably well approximated by

$$n = 3.16(k - 0.25)\ln\frac{m}{q^3}.$$

Appendix D

The capacity of convolution-based associative memories

The simplest type of convolution memory is one that stores pairs. In this appendix I analyze the capacity and scaling of this type of memory. In the first section I analyze the case where there is no systematic similarity among vectors. In the second section I analyze the case where there is some similarity. I will avoid duplicating derivations analogous to those in Appendix B.

D.1 A memory for paired-associates with no similarity among vectors

I use the following symbols and definitions:

- n, the vector dimension. "Random" vector have elements independently drawn from $N(0, 1/n)$.
- $\mathbb{E}$, a set of m random vectors.
- $\mathbf{t}$, a memory trace which is the (unnormalized) superposition of k distinct unordered pairs of vectors from $\mathbb{E}$, with the conditions that the two vectors in a pair must be different, and vectors in different pairs must be different:

$$\mathbf{t} = \sum_{i=1}^{k} \mathbf{x}_i \circledast \mathbf{y}_i, \quad \mathbf{x}_i, \mathbf{y}_i \in \mathbb{E}, \quad \mathbf{x}_i \neq \mathbf{y}_j \forall i, j, \quad \mathbf{x}_i \neq \mathbf{x}_j \forall i \neq j, \quad \mathbf{y}_i \neq \mathbf{y}_j \forall i \neq j$$

- Pr(All Correct), the probability of correctly determining which vectors are and are not stored in the memory trace.

In this analysis I assume that we do not know what the appropriate cues are, so to find all the pairs we must try every combination of cue and probe. In order to allow the retrieval process to be simple I assume that it tests for an identical cue and probe, even though no such pair is stored in the trace.

To test for the presence of a pair $(\mathbf{c}, \mathbf{p})$ in $\mathbf{t}$ we first decode it with the *cue* $\mathbf{c}$, and calculate the dot-product with the *probe* $\mathbf{p}$. This gives a scalar signal:

$$s = \mathbf{t} \circledast \mathbf{c}^* \cdot \mathbf{p}$$

Signal	Example	E	var	Number of tests	Description (referring to $\mathbf{t} \circledast \mathbf{c}^* \cdot \mathbf{p}$)
a	$\mathbf{t} \circledast \mathbf{x}_1{}^* \cdot \mathbf{y}_1$	1	$\frac{k+5}{n}$	k	$\mathbf{c} \neq \mathbf{p}$ and $(\mathbf{c}, \mathbf{p})$ is a pair in the trace.
$r_{1=}$	$\mathbf{t} \circledast \mathbf{x}_1^* \cdot \mathbf{x}_1$	0	$\frac{2k+4}{n}$	$2k$	$\mathbf{c} = \mathbf{p}$ and appears in the trace.
r_2	$\mathbf{t} \circledast \mathbf{x}_1^* \cdot \mathbf{x}_2$	0	$\frac{k+2}{n}$	$2k(k-1)$	$\mathbf{c} \neq \mathbf{p}$ and both $\mathbf{c}$ and $\mathbf{p}$ are in the trace (but are not members of the same pair.)
r_1	$\mathbf{t} \circledast \mathbf{x}_1^* \cdot \mathbf{z}$	0	$\frac{k+1}{n}$	$2k(m-2k)$	$\mathbf{c} \neq \mathbf{p}$ and one of them is in the trace.
$r_{0=}$	$\mathbf{t} \circledast \mathbf{z}^* \cdot \mathbf{z}$	0	$\frac{2k}{n}$	$m-2k$	$\mathbf{c} = \mathbf{p}$ and does not appear in the trace.
r_0	$\mathbf{t} \circledast \mathbf{z}^* \cdot \mathbf{w}$	0	$\frac{k}{n}$	$\frac{(m-2k)(m-2k-1)}{2}$	$\mathbf{c} \neq \mathbf{p}$ and neither of them are in the trace.

Table D.1: Means and variances of signals in a paired-associates convolution memory. The examples refer to the trace $\mathbf{t} = \sum_{i=1}^{k} \mathbf{x}_i \mathbf{y}_i$, and two vectors $\mathbf{z}$ and $\mathbf{w}$ from $\mathbb{E}$ not in the trace.

Calculating the dot-product of $\mathbf{t} \circledast \mathbf{c}^*$ with the probe corresponds to what a clean-up memory does, for each vector stored in it. Swapping the cue and probe gives identical results, as does probing the trace directly with $\mathbf{c} \circledast \mathbf{p}$, since

$$\mathbf{t} \circledast \mathbf{c}^* \cdot \mathbf{p} \;=\; \mathbf{t} \circledast \mathbf{p}^* \cdot \mathbf{c} \;=\; \mathbf{t} \cdot \mathbf{c} \circledast \mathbf{p}$$

are identities of convolution algebra.

There are five distributions of reject signals and one distribution of accept signals. The distribution of a reject signal depends on how many of the cue and probe occur in the trace (0, 1, or 2) and on whether the cue is equal to the probe. Table D.1 lists the six different signals, along with their means and variances, and the number of each of these signals that must be tested to probe exhaustively for each pair in the trace. The variances were calculated by adding the appropriate variances from Table 3.1, ignoring the terms of order $1/n^2$.

The derivations of all the signals are similar. I show the derivation of the signal (r_1) for when one of the cue and probe is in the trace. Without loss of generality I assume the cue is equal to $\mathbf{x}_1$, and the probe is $\mathbf{z}$, some element of *SetE* not in the trace.

$$\begin{aligned}
r_1 &= \mathbf{t} \circledast \mathbf{x}_1^* \cdot \mathbf{z} \\
&= \sum_{i=1}^{k} \mathbf{x}_i \circledast \mathbf{y}_i \circledast \mathbf{x}_1^* \cdot \mathbf{z} \\
\mathrm{E}[r_1] &= \mathrm{E}\left[\sum_{i=1}^{k} \mathbf{x}_i \circledast \mathbf{y}_i \circledast \mathbf{x}_1^* \cdot \mathbf{z}\right] \\
&= \sum_{i=1}^{k} \mathrm{E}[\mathbf{x}_i \circledast \mathbf{y}_i \circledast \mathbf{x}_1^* \cdot \mathbf{z}] \\
&= \sum_{i=1}^{k} 0 = 0 \\
\mathrm{var}[r_1] &= \mathrm{var}\left[\sum_{i=1}^{k} \mathbf{x}_i \circledast \mathbf{y}_i \circledast \mathbf{x}_1^* \cdot \mathbf{z}\right]
\end{aligned}$$

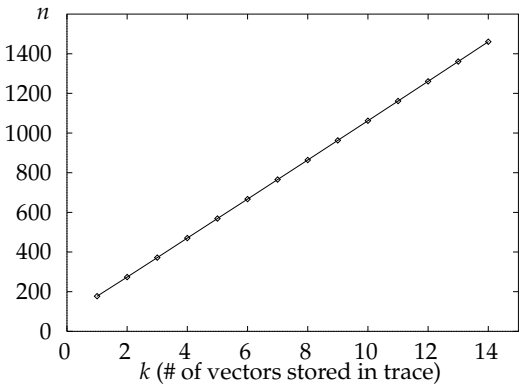


Figure D.1: Scaling of n with k for Pr(Error) = 0.01 and m = 1000 in a paired-associates convolution memory.

$$= \text{var}[\mathbf{x}_1 \circledast \mathbf{y}_1 \circledast \mathbf{x}_1^* \cdot \mathbf{z}] + \sum_{i=2}^{k} \text{var}[\mathbf{x}_i \circledast \mathbf{y}_i \circledast \mathbf{x}_1^* \cdot \mathbf{z}]$$

$$= \frac{2}{n} + k\frac{1}{n} = \frac{k+1}{n}$$

The variance of the sum is equal to the sum of the variance because the covariance between all the terms in the sum is zero (e.g., $\text{cov}[\mathbf{x}_1 \circledast \mathbf{y}_1 \circledast \mathbf{x}_1^* \cdot \mathbf{z}, \mathbf{x}_2 \circledast \mathbf{y}_2 \circledast \mathbf{x}_1^* \cdot \mathbf{z}] = 0$).[1] This is true for the sums of variances for all the signals in this scenario.

Since there are $2k(n-2k)$ cue-probe pairs that will generate a signal from this distribution, the r_1 signal must be rejected this many times.

For a threshold t the probability of correctly identifying all the pairs in the trace is:

$$\begin{aligned}\text{Pr(All Correct)} &= \text{Pr}(a > t)^k + \text{Pr}(r_{1=} < t)^{2k} \\ &\quad + \text{Pr}(r_2 < t)^{2k(k-1)} + \text{Pr}(r_1 < t)^{2k(m-2k)} \\ &\quad + \text{Pr}(r_{0=} < t)^{m-2k} + \text{Pr}(r_0 < t)^{(m-2k)(m-2k-1)/2} \end{aligned} \quad \text{(D.1)}$$

The scaling of k with n to give a constant Pr(All Correct) is slightly less than linear. It is shown in Figure D.1. This scaling is of the same nature as that of the simple superposition memory, for the same reasons. Likewise, the scaling of Pr(Error) and m with n is exponential.

[1] This would not be true if pairs of identical vectors were allowed in the trace, since $\text{cov}[\mathbf{x}_2 \circledast \mathbf{x}_2 \circledast \mathbf{x}_1^* \cdot \mathbf{z}, \mathbf{x}_3 \circledast \mathbf{x}_3 \circledast \mathbf{x}_1^* \cdot \mathbf{z}] \neq 0$.

D.2 A memory for paired-associates with some similarity among vectors

I now consider a convolution memory used to store variable-value bindings, where there is similarity among some of the values. In order to avoid complicating the analysis, I assume variables and values come from different sets.

The following symbols are used:

- n, the vector dimension. "Random" vector have elements independently drawn from $N(0, 1/n)$.
- $\mathbb{E}_p$, a set of m_p value-vectors with average similarity α^2. They have the general form $\mathbf{p}_i = \alpha\mathbf{p} + \beta\mathbf{d}_i$, where $\mathbf{p}$ is the component the vectors in $\mathbb{E}_p$ have in common, $\mathbf{d}_i$ is a random vector, and the scalars α and β are positive weighting constants such that $\alpha^2 + \beta^2 = 1$ (so that $E[|\mathbf{p}_i|] = 1$). $\mathbf{p}$ is a random vector. In contrast with Appendix B, I do not assume that $|\mathbf{p}| = 1$.
- $\mathbb{E}$, the set of m value-vectors, containing $\mathbb{E}_p$ and $(m - m_p)$ random vectors. The set $\{\mathbb{E} - \mathbb{E}_p\}$ is denoted by $\mathbb{E}_{-p}$.
- $\mathbb{V}$, a set of v variable vectors, all of which are random.
- $\mathbf{t}$, a trace with k variable-value bindings for similar values (from $\mathbb{E}_p$), and j variable-value bindings for dissimilar values (from $\mathbb{E}_{-p}$):

$$\mathbf{t} = \underbrace{\mathbf{x}_1 \circledast \mathbf{p}_1 + \mathbf{x}_2 \circledast \mathbf{p}_2 + \mathbf{x}_3 \circledast \mathbf{p}_3 + \cdots}_{\text{Similar values } (k \text{ pairs})} + \underbrace{\mathbf{x}_{k+1} \circledast \mathbf{y}_1 + \mathbf{x}_{k+2} \circledast \mathbf{y}_2 + \cdots}_{\text{Non-similar values } (j \text{ pairs})},$$

  where $\mathbf{x}_i \in \mathbb{V}$, $\mathbf{p}_i \in \mathbb{E}_p$, and $\mathbf{y}_i \in \mathbb{E}_{-p}$. No value or variable appears more than once in the bindings in the trace.

Variables are unbound by using the variable as the cue and the possible values as the probes. To find out which value is associated with a variable $\mathbf{x}$ in the trace $\mathbf{t}$ we compute $\mathbf{t} \circledast \mathbf{x}^*$ and compare (using dot-product) the result to each of the vectors in $\mathbb{E}$. It is also possible to use a value as a probe to discover which variable it is bound to.

For the purposes of calculating the distributions of accept and reject signals, there are four classes of values and three classes of variables. These are explained in Table D.2. The example vectors in this table correspond to the vectors in the typical trace just mentioned, assuming that $\mathbf{p}_0$ is some value-vector in $\mathbb{E}_p$ that is not in the trace, $\mathbf{z}$ is some value-vector in $\mathbb{E}_{-p}$ that is not in the trace, and $\mathbf{x}_0$ is some variable-vector that is not in the trace. A signal is generated by a cue (variable) and a probe (value). Consequently there are two distributions of accept signals and twelve distributions of reject signals. These signals are their relationship to the classes are illustrated in Figure D.2. The rows are the variable classes and the columns are the value classes. The means and variances, and the number of instances of each of the fourteen signals are shown in Table D.3.

If nothing is known about what variables or values are stored in the trace, then the probability of correctly identifying all the variables and values is the probability that all

Class	Value	E.g.	Variable	E.g.
1	$\in \mathbb{E}_p$ in the trace	$\mathbf{p}_1$		$\mathbf{x}_1$
2	$\in \mathbb{E}_p$ not in the trace	$\mathbf{p}_0$	N/A	
3	$\in \mathbb{E}_{-p}$ in the trace	$\mathbf{y}_1$		$\mathbf{x}_{j+1}$
4	$\in \mathbb{E}_{-p}$ not in the trace	$\mathbf{z}$	not bound	$\mathbf{x}_0$

Table D.2: Classes of values and variables in a variable-binding memory with similarity. The class of a variable is determined by which (if any) value it is bound to. The example vectors refer to the typical trace shown above.

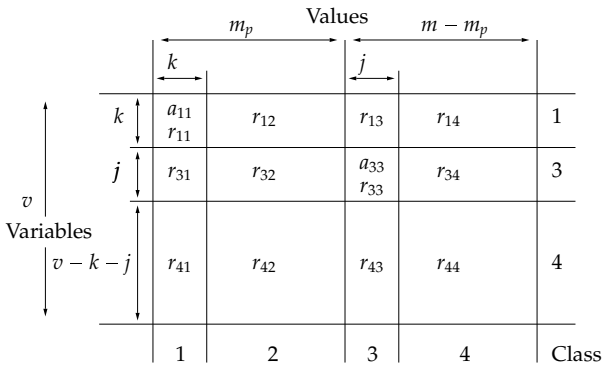


Figure D.2: The different signals in a variable-binding memory with similarity. The numbers of variables and values are shown beside the arrows. Each square is labelled with the name of the signal distribution which results from using a cue and probe from that row and column.

reject signals are less than the threshold and all accept signals are greater than the threshold. The signals can be assumed to be independent as in the previous models.[2]

The scaling of n with k and j, the numbers of items stored in the trace, is shown in Figure D.3. The three steeper lines are for n versus k, the number of similar values stored. The three lines are for $j = 0$, 1, and 2. The three less steep lines are for n versus j, the number of non-similar values stored. The gradient for k is steeper because higher dimensionality is needed to discriminate similar vectors. Note that in all cases the relationship between n and k (or j) is near linear, just the gradient differs.

The scaling of n with α and k is shown in Figure D.4. As in the superposition memory model with similarity, the capacity is relatively unaffected by small values of α.

[2]This assumption is not entirely correct, as in any particular model there will be some correlations due to the similarity among the members of $\mathbb{E}_p$, e.g., $\mathbf{x}_1 \circledast \mathbf{p}_1 \circledast \mathbf{x}_1^* \cdot \mathbf{p}_2$ will be correlated with $\mathbf{x}_1 \circledast \mathbf{p}_1 \circledast \mathbf{x}_1^* \cdot \mathbf{p}_3$. This leads to a slight underestimation of Pr(All Correct), because the expected number of errors is correct, but errors will tend to occur in clusters. Experiments give similar Pr(All Correct) to those predicted by the model. For example, in 10,000 trials with $n = 1024$, $m = 200$, $m_p = 100$, $v = 200$, $k = 3$, $j = 1$, $\alpha^2 = 0.5$, for 10 different sets $\mathbb{E}$, there were a total of 699 trials in which not all elements of the trace were correctly identified. The above analysis gives Pr(All Correct) = 0.907, which predicts 930 failures (with standard deviation 9.2) out of 10,000 trials.

Signal	Mean	Variance	Number
a_{11}	1	$\frac{1}{n}(5 + k + j + (k-1)\alpha^4)$	k
a_{33}	1	$\frac{1}{n}(5 + k + j)$	j
r_{11}	α^2	$\frac{1}{n}(j + (k+2)(1+\alpha^4))$	$k(k-1)$
r_{12}	α^2	$\frac{1}{n}((3+k)(1+\alpha^4) + j - 2)$	$k(m_p - k)$
r_{13}	0	$\frac{1}{n}(k + j + 2)$	kj
r_{14}	0	$\frac{1}{n}(k + j + 1)$	$k(m - m_p - j)$
r_{31}	0	$\frac{1}{n}((k-1)(1+\alpha^4) + j + 3)$	jk
r_{32}	0	$\frac{1}{n}(k(1+\alpha^4) + j + 1)$	$j(m_p - k)$
r_{33}	0	$\frac{1}{n}(k + j + 2)$	$j(j-1)$
r_{34}	0	$\frac{1}{n}(k + j + 1)$	$j(m - m_p - j)$
r_{41}	0	$\frac{1}{n}((k-1)(1+\alpha^4) + j + 2)$	$(v - j - k)k$
r_{42}	0	$\frac{1}{n}(k(1+\alpha^4) + j)$	$(v - j - k)(m_p - k)$
r_{43}	0	$\frac{1}{n}(k + j + 1)$	$(v - j - k)j$
r_{44}	0	$\frac{1}{n}(k + j)$	$(v - j - k)(m - m_p - j)$

Table D.3: Means, variances, and numbers of signals in a variable-binding memory with similarity.

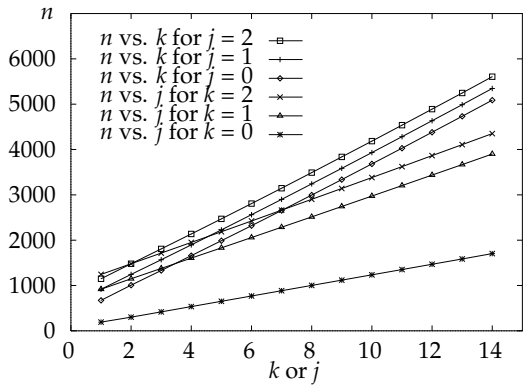


Figure D.3: Scaling of n with k and j for Pr(Error) = 0.01, $m = 100,000$, $v = 100,000$, $m_p = 100$, and $\alpha^2 = \beta^2 = 0.5$ in a variable-binding memory with similarity.

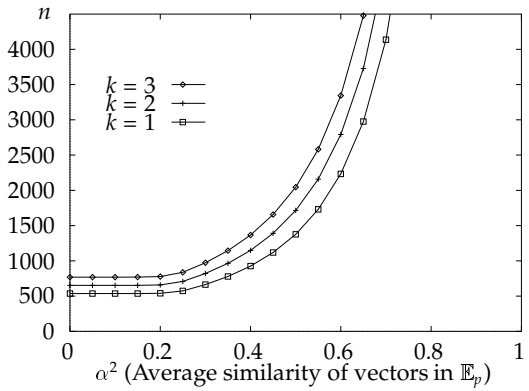


Figure D.4: Scaling of n with α for various values of k for Pr(Error) = 0.01, $m = 100,000$, $v = 100,000$, $m_p = 100$, and $j = 3$ in a variable-binding memory with similarity.

Figure D.5 shows the scaling of n with the number of similar items in the clean-up memory. As with the previous scenarios, the number of items in clean-up memory (similar items in this case) can increase exponentially with the vector dimension.

Figure D.6 shows the scaling of n with the number of non-similar items in the clean-up memory. This graph shows how the number of similar items dominates the capacity equation: n is nearly constant for less than 10^{14} non-similar items in memory. Beyond that, the number of non-similar items can increase exponentially with the vector dimension.

D.3 Comments on the variable-binding memory with similarity

Unlike in a simple superposition memory for similar vectors, the similar value-vectors stored in the variable binding memory do not interfere with means of the accept and reject signals. This is due to the "randomizing" property of convolving with a random variable; $\mathbf{x}_1 \circledast \mathbf{a}$ and $\mathbf{x}_2 \circledast \mathbf{a}$ have an expected similarity of zero if $\mathbf{x}_1$ and $\mathbf{x}_2$ are random. Any combination of similar and non-similar values can be bound without affecting the means of signals. The similarity among values only acts to increase the signal variances. However, this is not the case when there is similarity among the variable vectors used in the trace. Furthermore, the signal variances will increase dramatically if the variables are similar to the values (due to the high variance of $\mathbf{x} \circledast \mathbf{x} \cdot \mathbf{x} \circledast \mathbf{x}$).

It is possible to associate the same value with two different variables. E.g., $\mathbf{t} = \mathbf{x}_1 \circledast \mathbf{y}_1 + \mathbf{x}_2 \circledast \mathbf{y}_2$. The same process as before can be used to find which value is bound to a give variable. The signal variances will be marginally higher, but the means will be the same.

If the same variable is associated with two different values, then correlating with that variable will give a blend of the two values. E.g. $(\mathbf{x}_1 \circledast \mathbf{y}_1 + \mathbf{x}_1 \circledast \mathbf{y}_2) \circledast \mathbf{x}_1^*$ will give a noisy version of $\mathbf{a} + \mathbf{b}$. This can be treated it as a superposition memory trace.

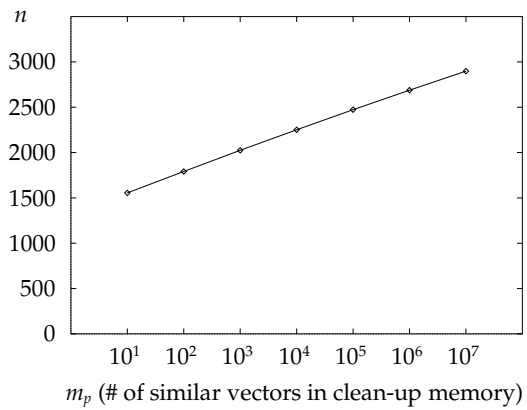


Figure D.5: Scaling of n with m_p (the number of similar items in memory) for Pr(Error) = 0.01, $m = 10^8$, $v = 100,000$, $k = 3$, $j = 3$, and $\alpha^2 = \beta^2 = 0.3$ in a variable-binding memory with similarity.

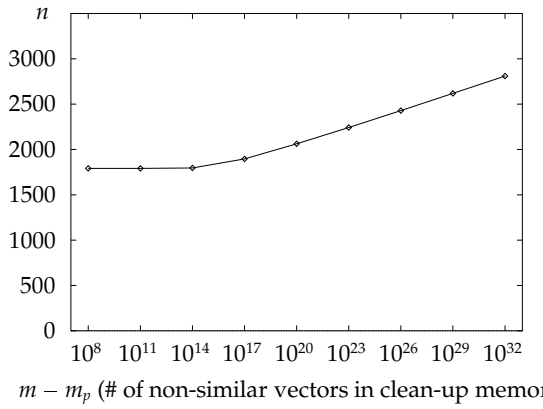


Figure D.6: Scaling of n with $m - m_p$ (the number of non-similar items in memory) for Pr(Error) = 0.01, $m_p = 100$, $k = 3$, $j = 3$, and $\alpha^2 = \beta^2 = 0.3$ in a variable-binding memory with similarity.

Appendix E

A lower bound for the capacity of convolution memories

In this appendix I give an analytic lower bound on the capacity of a convolution memory in the case where there is not any systematic similarity among vectors.

I use the following symbols and definitions (which are the same as in Section D.1):

- n, the vector dimension. "Random" vector have elements independently drawn from $N(0, 1/n)$.
- $\mathbb{E}$, a set of m random vectors.
- $\mathbf{t}$, a memory trace which is the (unnormalized) superposition of k distinct unordered pairs of vectors from $\mathbb{E}$, with the conditions that the two vectors in a pair must be different, and vectors in different pairs must be different:

$$\mathbf{t} = \sum_{i=1}^{k} \mathbf{x}_i \circledast \mathbf{y}_i, \quad \mathbf{x}_i, \mathbf{y}_i \in \mathbb{E}, \quad \mathbf{x}_i \neq \mathbf{y}_j \forall i, j, \quad \mathbf{x}_i \neq \mathbf{x}_j \forall i \neq j, \quad \mathbf{y}_i \neq \mathbf{y}_j \forall i \neq j$$

- Pr(All Correct), the probability of correctly determining which vectors are and are not stored in the memory trace.

Equation D.1 from Section D.1 gives the probability of correctly identifying all the pairs in the trace:

$$\begin{aligned} \Pr(\text{All Correct}) \;=\; & \Pr(a > t)^k + \Pr(r_{1=} < t)^{2k} \\ & + \Pr(r_2 < t)^{2k(k-1)} + \Pr(r_1 < t)^{2k(m-2k)} \\ & + \Pr(r_{0=} < t)^{m-2k} + \Pr(r_0 < t)^{(m-2k)(m-2k-1)/2} \end{aligned}$$

Using the same inequalities as in Appendix C we get:

$$\begin{aligned} q \;&<\; \frac{m(m+1)}{2}\,\text{tail}\left(\frac{1}{2}\sqrt{\frac{2k+4)}{n}}\right) \\ &<\; m^2\, e^{\frac{-n}{16(k+2)}} \qquad \text{if } n > \frac{4(k+2)}{\pi} \\ \text{or} \quad n \;&<\; 16(k+2)\ln\frac{m^2}{q} \qquad \text{if } n > \frac{4(k+2)}{\pi} \end{aligned}$$

Numerical solutions of the Equation D.1 for k in the range (2..14), m in $(10^2 \ldots 10^{10})$, and q in $(10^{-2} \ldots 10^{-10})$ are reasonably well approximated by

$$n = 4.5(k + 0.7)\ln\frac{m}{30q^4}.$$

Appendix F

Means and variances of a signal

The detailed calculation of the mean and variance one of the components of one of the signals in Section 3.10.4 is presented in this appendix.

Expanding the signal X_{mark} gives

$$\begin{aligned}
X_{mark} &= \mathbf{s}_1 \circledast \mathbf{eat}_{agt}{}^* \cdot \frac{1}{\sqrt{3}}\mathbf{id}_{mark} \\
&= \frac{1}{\sqrt{3}}(\mathbf{eat} + \mathbf{eat}_{agt} \circledast \mathbf{mark} + \mathbf{eat}_{obj} \circledast \mathbf{the_fish}) \\
&\quad \circledast \mathbf{eat}_{agt}{}^* \cdot \frac{1}{\sqrt{3}}\mathbf{id}_{mark} \\
&= \frac{1}{3}(\mathbf{eat} + \mathbf{eat}_{agt} \circledast \frac{1}{\sqrt{3}}(\mathbf{being} + \mathbf{person} + \mathbf{id}_{mark}) \\
&\quad + \mathbf{eat}_{obj} \circledast \mathbf{the_fish}) \circledast \mathbf{eat}_{agt}{}^* \cdot \mathbf{id}_{mark} \\
&= \frac{1}{3}\mathbf{eat} \circledast \mathbf{eat}_{agt}{}^* \cdot \mathbf{id}_{mark} \\
&\quad + \frac{1}{3\sqrt{3}}\mathbf{eat}_{agt} \circledast \mathbf{being} \circledast \mathbf{eat}_{agt}{}^* \cdot \mathbf{id}_{mark} \\
&\quad + \frac{1}{3\sqrt{3}}\mathbf{eat}_{agt} \circledast \mathbf{person} \circledast \mathbf{eat}_{agt}{}^* \cdot \mathbf{id}_{mark} \\
&\quad + \frac{1}{3\sqrt{3}}\mathbf{eat}_{agt} \circledast \mathbf{id}_{mark} \circledast \mathbf{eat}_{agt}{}^* \cdot \mathbf{id}_{mark} \\
&\quad + \frac{1}{3}\mathbf{eat}_{obj} \circledast \mathbf{the_fish} \circledast \mathbf{eat}_{agt}{}^* \cdot \mathbf{id}_{mark}
\end{aligned}$$

The expectations and variances of these terms can be found by consulting Table 3.1. It is not necessary to expand the vectors $\mathbf{eat}_{agt}$, $\mathbf{eat}_{obj}$ or **the_fish**, as the components of these are independent of the other vectors appearing in the same terms. The expectation of the fourth term is $\frac{1}{3\sqrt{3}}$ (row 4 in Table 3.1), and all the expectation of the remaining terms is zero. These five terms are independent and thus the variance of the sum is the sum of the variances. The variance of the first term is $\frac{1}{9n}$ (row 3 in Table 3.1), the variance of the second and third terms is $\frac{2n+2}{27n^2}$ (row 8), the variance of the fourth term is $\frac{6n+4}{27n^2}$ (row 6), and the variance of the fifth term is $\frac{1}{9n}$ (row 10). These terms are uncorrelated, so their expectations

and variances can be summed to give

$$\mathrm{E}[X_{mark}] = \frac{1}{3\sqrt{3}}, \quad \mathrm{var}[X_{mark}] = \frac{16n+8}{27n^2} \approx 0.593/n.$$

The expectations and variances of X_p and Z can be calculated in a similar manner. They are:

$$\mathrm{E}[X_p] = 0, \quad \mathrm{var}[X_p] = \frac{12n+6}{27n^2} \approx 0.444/n$$

$$\mathrm{E}[Z] = \frac{2}{3\sqrt{3}}, \quad \mathrm{var}[Z] = \frac{34n+20}{27n^2} \approx 1.26/n$$

Appendix G

The effect of normalization on dot-products

As discussed in Section 3.5.3, it is usually preferable to normalize all vectors so that their magnitude is 1. However, the means and variances of dot-products reported in Table 3.1 in Section 3.6.1 do not apply when vectors are normalized. With normalized vectors, means of dot-products are generally the same or slightly lower, and variances are the same or much lower.

Recall that $\langle \mathbf{x} \rangle$ denotes the normalized version of $\mathbf{x}$ (so that $\langle \mathbf{x} \rangle \cdot \langle \mathbf{x} \rangle = 1$):

$$\langle \mathbf{x} \rangle = \frac{1}{\sqrt{\sum_{i=0}^{n-1} x_i}} \mathbf{x}$$

For the purposes of this section, a *random* vector of dimension n is one whose elements are chosen randomly from $N(0, 1/n)$.

G.1 Means and variances of dot-products of vectors with varying similarity

Consider the first two rows in Table 3.1, for $\mathbf{x} \cdot \mathbf{x}$ and $\mathbf{x} \cdot \mathbf{y}$, where $\mathbf{x}$ and $\mathbf{y}$ have elements independently drawn from $N(0, 1/n)$. For scaled vectors the variances are:

$$\text{var}[\mathbf{x} \cdot \mathbf{x}] = 2/n \quad \text{and} \quad \text{var}[\mathbf{x} \cdot \mathbf{y}] = 1/n.$$

However, if we normalize the vectors, the variances are:[1]

$$\text{var}[\langle \mathbf{x} \rangle \cdot \langle \mathbf{x} \rangle] = 0 \quad \text{and} \quad \text{var}[\langle \mathbf{x} \rangle \cdot \langle \mathbf{y} \rangle] = 1/n.$$

Using these two rows in Table 3.1 we can calculate the expected value and variance of the dot-product of $\mathbf{x}$ with vectors which have varying similarity to $\mathbf{x}$ (if normalization is not used). For example, for $d = \mathbf{x} \cdot 1/\sqrt{3}(\mathbf{x} + \mathbf{a} + \mathbf{b})$ (where $\mathbf{x}$, $\mathbf{a}$, and $\mathbf{b}$ are random vectors) we have:

$$\text{E}[d] = 1/\sqrt{3} \quad \text{and} \quad \text{var}[d] = 4/3n.$$

[1] I am indebted to Radford Neal for a proof that $\text{var}[\langle \mathbf{x} \rangle \cdot \langle \mathbf{y} \rangle] = 1/n$.

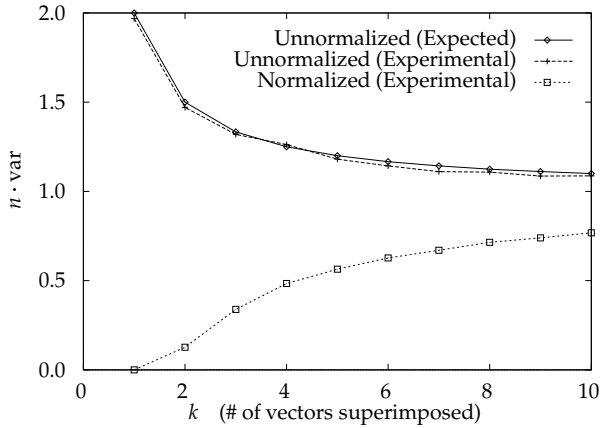


Figure G.1: Variances of the dot-product of $\mathbf{x}$ with superpositions of $\mathbf{x}$ with (k-1) other vectors. The experimental data are for $n = 1024$ and 10,000 runs.

The $1/\sqrt{3}$ scale factor makes the expected magnitude of $1/\sqrt{3}(\mathbf{x} + \mathbf{a} + \mathbf{b})$ equal to 1. In general, for

$$d_k = \tfrac{1}{\sqrt{k}}(\mathbf{x} + \textstyle\sum_{i=1}^{k-1} \mathbf{y}_i) \cdot \mathbf{x}$$

(where $\mathbf{x}$ and $\mathbf{y}_i$ are random vectors) we have

$$\text{E}[d_k] = \frac{1}{\sqrt{k}} \quad \text{and} \quad \text{var}[d_k] = \frac{1}{n}\frac{k+1}{k}.$$

G.1.1 Experimentally observed means and variances of dot-products

The expected and experimentally observed variances for dot-products of scaled vectors (d) are shown in Figure G.1. As expected, these are almost identical. The experimentally observed variances for dot-products of normalized vectors (d') are also shown:

$$d'_k = \left(\langle \langle \mathbf{x} \rangle + \textstyle\sum_{i=1}^{k-1} \langle \mathbf{y}_i \rangle \rangle \right) \cdot \langle \mathbf{x} \rangle,$$

These variances are considerably lower, though they tend to the same limit (1) as k tends to infinity. The experiments were done with $n = 1024$ and 10,000 runs.

The experimentally observed means for dot-products of scaled and normalized vectors were indistinguishable from the expected values (i.e., $1/\sqrt{k}$).

G.2 Means and variances of dot-products of convolution expressions

Rows 3 through 10 in Table 3.1 concern the means and variances of dot-products that arise when convolution bindings are decoded or compared.

	Unnormalized Expected (Analytical)		Unnormalized Experimental		Normalized Experimental Decoding		Normalized Experimental Similarity	
Expression	mean	$n \cdot$ var	mean	$n \cdot$ var	mean	$n \cdot$ var	mean	$n \cdot$ var
(1) $\mathbf{a} \cdot \mathbf{a}$	1	2	1.0003	2.0094	1.0000	0.0000	1.0000	0.0000
(2) $\mathbf{a} \cdot \mathbf{b}$	0	1	−0.0007	1.0251	0.0002	0.9952	0.0000	0.9842
(3) $\mathbf{a} \cdot \mathbf{a} \circledast \mathbf{b}$	0	2.0001	−0.0002	2.0122	−0.0001	1.9897	−0.0002	2.0480
(4) $\mathbf{a} \cdot \mathbf{b} \circledast \mathbf{c}$	0	1	0.0006	1.0014	−0.0004	0.9995	−0.0001	0.9926
(5) $\mathbf{a} \circledast \mathbf{a} \circledast \mathbf{a}^* \cdot \mathbf{a}$	2.0020	40.109	2.0038	39.9667	1.0000	0.0000	0.8201	0.5014
(6) $\mathbf{a} \circledast \mathbf{b} \circledast \mathbf{a}^* \cdot \mathbf{b}$	1	6.0039	0.9998	5.9699	1.0000	0.0000	0.7088	0.4524
(7) $\mathbf{a} \circledast \mathbf{b} \circledast \mathbf{a}^* \cdot \mathbf{a}$	0	6.0176	−0.0015	6.1133	0.0003	2.9795	0.0002	2.9631
(8) $\mathbf{a} \circledast \mathbf{b} \circledast \mathbf{a}^* \cdot \mathbf{c}$	0	2.0020	−0.0003	2.0184	−0.0001	1.9378	−0.0001	0.9974
(9) $\mathbf{a} \circledast \mathbf{b} \circledast \mathbf{c}^* \cdot \mathbf{c}$	0	2.0020	−0.0004	1.9866	0.0000	1.0049	0.0006	1.9434
(10) $\mathbf{a} \circledast \mathbf{b} \circledast \mathbf{c}^* \cdot \mathbf{d}$	0	1	0.0006	1.0073	−0.0001	1.0135	0.0003	1.0149

Table G.1: Experimentally observed means and variances of dot-products of common convolution expressions, for normalized and unnormalized versions. In the normalized version, normalization is performed at every step. For example, for (10), the expression for normalized decoding is $\langle\langle\langle \mathbf{a} \rangle \circledast \langle \mathbf{b} \rangle\rangle \circledast \langle \mathbf{c} \rangle^*\rangle \cdot \langle \mathbf{d} \rangle$ and the expression for normalized similarity is $\langle\langle \mathbf{a} \rangle \circledast \langle \mathbf{b} \rangle\rangle \cdot \langle\langle \mathbf{c} \rangle \circledast \langle \mathbf{d} \rangle\rangle$. The statistics are over 10,000 runs, and $n = 1024$. The variances are multiplied by n to make them easier to read and compare.

The experimentally observed means and variances for various dot-products of convolution products are shown in Table G.1. The means and variances for the decoding and similarity expressions are reported separately, since the decoding/similarity identity

$$\mathbf{a} \circledast \mathbf{b} \circledast \mathbf{c}^* \cdot \mathbf{d} \;=\; \mathbf{a} \circledast \mathbf{b} \cdot \mathbf{c} \circledast \mathbf{d}$$

does not hold if vectors are normalized. It turns out that in general

$$\langle\langle\langle \mathbf{a} \rangle \circledast \langle \mathbf{b} \rangle\rangle \circledast \langle \mathbf{c} \rangle^*\rangle \cdot \langle \mathbf{d} \rangle \;\neq\; \langle\langle \mathbf{a} \rangle \circledast \langle \mathbf{b} \rangle\rangle \cdot \langle\langle \mathbf{c} \rangle \circledast \langle \mathbf{d} \rangle\rangle,$$

and the expected values are the same only when they are zero.

The variances are significantly lower with normalization. The non-zero means for decoding convolution bindings with normalization are also lower. For example, the sample mean of $\langle\langle\langle \mathbf{a} \rangle \circledast \langle \mathbf{b} \rangle\rangle \circledast \langle \mathbf{a} \rangle^*\rangle \cdot \langle \mathbf{b} \rangle$ is 0.7088, whereas without normalization the mean of this is 1. However, the decrease in variance outweights the decrease in means – performance is still likely to be superior when normalization is used.

Appendix H

HRRs with circular vectors

In this appendix, I present analytic and experimental results concerning the means and variances for dot-products of superpositions and bindings in the *circular* system described in Chapter 4. I also compare the circular system with unnormalized and normalized versions of the standard system.

H.1 Means and variances of dot-products of vectors with varying similarity

For the standard system, it is easy to show that the similarity of $\mathbf{x}$ and the scaled (but not normalized) superposition of $\mathbf{x}$ with $k-1$ other vectors $(1/\sqrt{k+1}(\mathbf{x} + \sum_{i=1}^{k-1} \mathbf{y}_i))$ is $1/\sqrt{k}$. This remains true if vectors are normalized rather than scaled. Variances are also easily calculated, at least in the unnormalized version. For circular vectors, these calculations are more difficult. I present simulation results for $k = 1 \ldots 10$ and analytic derivations for means and variances in the cases of $k = 0$ and $k = 1$.

Figures H.1 and H.2 show the experimentally observed means and variances of

$$s_c = \bar{\theta} \cdot \mathrm{sp}(\bar{\theta}, \bar{\phi}_1, \ldots \bar{\phi}_{k-1})$$

for $k = 0$ to 10, where $\bar{\theta}$ and the $\bar{\phi}_i$ are random vectors. For comparison, the corresponding observations for unnormalized and normalized standard vectors are also plotted. These observations are for

$$s_u = \mathbf{x} \cdot \sqrt{\frac{1}{k}}(\mathbf{x} + \sum_{i=1}^{k-1} \mathbf{y}_i)$$

and

$$s_n = \langle \mathbf{x} \rangle \cdot \langle \langle \mathbf{x} \rangle + \sum_{i=1}^{k-1} \langle \mathbf{y}_i \rangle \rangle$$

where $\mathbf{x}$ and the $\mathbf{y}_i$ are random vectors.

For the analytic derivations I only consider single angles rather than a vector of angles. It is clear that the variance of a dot-product of circular vectors is proportional to $1/n$, because it is a sum of n independent random variables, multiplied by $1/n$. For the same reason, the expected value of the dot-product does not vary with n.

First, consider the dot-product of dissimilar angles: θ and ϕ independently distributed as $\mathrm{U}(-\pi, \pi)$. We can substitute $\psi \stackrel{d}{=} \mathrm{U}(-\pi, \pi)$ for $(\theta - \phi \bmod 2\pi)$ because the latter is distributed uniformly in $(-\pi, \pi]$.

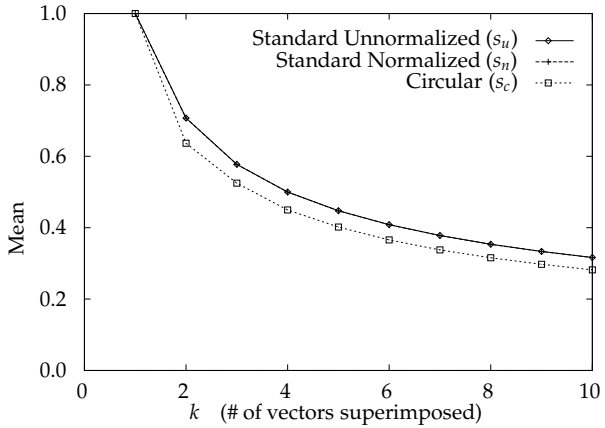


Figure H.1: Means of the dot-product of a random vector with superpositions of it and and $(k-1)$ other random vectors. The data are for $n = 512$ and $10,000$ runs. The results for the normalized and unnormalized systems overlap on this plot.

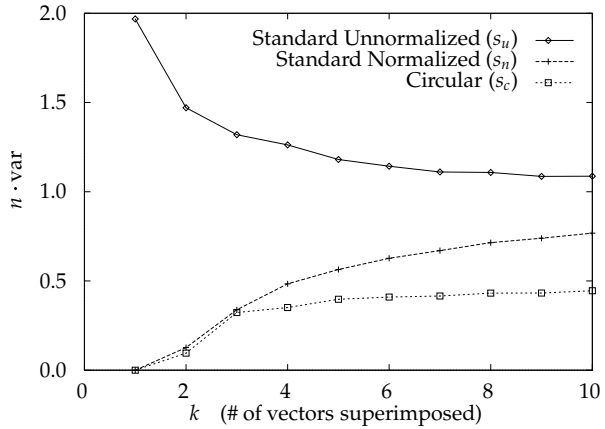


Figure H.2: Variances of the dot-product of a random vector with superpositions of it and $(k-1)$ other vectors. The data are for $n = 512$ and $10,000$ runs.

$$
\begin{aligned}
\mathrm{E}[\theta \cdot \phi] &= \mathrm{E}[\cos(\theta - \phi)] \\
&= \mathrm{E}[\cos(\psi)] \qquad \text{for } \psi \stackrel{d}{=} \mathrm{U}(-\pi, \pi) \\
&= \int_{-\pi}^{\pi} \frac{2}{\pi} \cos(\psi) d\psi \\
&= 0 \\
\mathrm{var}[\theta \cdot \phi] &= \mathrm{E}[\cos^2(\theta - \phi)] \\
&= \mathrm{E}[\cos^2(\psi)] \qquad \text{for } \psi \stackrel{d}{=} \mathrm{U}(-\pi, \pi) \\
&= \int_{-\pi}^{\pi} \frac{2}{\pi} \cos^2(\psi) d\psi \\
&= \frac{1}{2}
\end{aligned}
$$

Next consider the dot-product of an angle with itself (i.e., $k = 0$ in the graphs). Since $\cos(0) = 1$, we have

$$
\begin{aligned}
\mathrm{E}[\theta \odot \theta] &= 1 \\
\mathrm{var}[\theta \odot \theta] &= 0
\end{aligned}
$$

Finally consider the dot-product of θ with $\theta \oplus \phi$ (i.e., $k = 1$ in the graphs). The experimentally observered means and variances are shown in parenthesis.

$$
\begin{aligned}
\mathrm{E}[\theta \cdot (\theta \oplus \phi)] &= \mathrm{E}[\cos(\theta - (\theta \oplus \phi))] \\
&= \mathrm{E}[\cos(\phi/2)] \\
&= \int_{-\pi}^{\pi} \frac{2}{\pi} \cos(\phi/2) d\phi \\
&= \frac{2}{\pi} \quad = \quad 0.63662\ldots \quad (0.6365) \\
\mathrm{var}[\theta \cdot (\theta \oplus \phi)] &= \mathrm{E}[(\frac{2}{\pi} - \cos(\theta - (\theta \oplus \phi)))^2] \\
&= \mathrm{E}[(\frac{2}{\pi} - \cos(\phi/2))^2] \\
&= \int_{-\pi}^{\pi} \frac{2}{\pi} (\frac{2}{\pi} - \cos(\phi/2))^2 d\phi \\
&= \frac{-8 + \pi^2}{2\pi^2} \quad = \quad 0.094715\ldots \quad (0.09552)
\end{aligned}
$$

H.2 Means and variances of dot-products for similarity and decoding

The means and variances of dot-products for similarity and decoding are easy to calculate, especially in cases where no superposition is involved. Consider the various dot-products

	Circular Decoding and Similarity				Normalized Decoding		Normalized Similarity	
	Analytic		Experimental		Experimental		Experimental	
Expression	mean	$n \cdot$ var	mean	$n \cdot$ var	mean	$n \cdot$ var	mean	$n \cdot$ var
(1) $\mathbf{a} \cdot \mathbf{a}$	1	0	1.0000	0.0000	1.0000	0.0000	1.0000	0.0000
(2) $\mathbf{a} \cdot \mathbf{b}$	0	1/2	0.0000	0.5003	0.0002	0.9952	0.0000	0.9842
(3) $\mathbf{a} \cdot \mathbf{a} \odot \mathbf{b}$	0	1/2	0.0006	0.5028	−0.0001	1.9897	−0.0002	2.0480
(4) $\mathbf{a} \cdot \mathbf{b} \odot \mathbf{c}$	0	1/2	−0.0004	0.4989	−0.0004	0.9995	−0.0001	0.9926
(5) $\mathbf{a} \odot \mathbf{a} \odot \mathbf{a}^* \cdot \mathbf{a}$	1	0	1.0000	0.0000	1.0000	0.0000	0.8201	0.5014
(6) $\mathbf{a} \odot \mathbf{b} \odot \mathbf{a}^* \cdot \mathbf{b}$	1	0	1.0000	0.0000	1.0000	0.0000	0.7088	0.4524
(7) $\mathbf{a} \odot \mathbf{b} \odot \mathbf{a}^* \cdot \mathbf{a}$	0	1/2	0.0000	0.5003	0.0003	2.9795	0.0002	2.9631
(8) $\mathbf{a} \odot \mathbf{b} \odot \mathbf{a}^* \cdot \mathbf{c}$	0	1/2	0.0005	0.4903	−0.0001	1.9378	−0.0001	0.9974
(9) $\mathbf{a} \odot \mathbf{b} \odot \mathbf{c}^* \cdot \mathbf{c}$	0	1/2	0.0001	0.4998	0.0000	1.0049	0.0006	1.9434
(10) $\mathbf{a} \odot \mathbf{b} \odot \mathbf{c}^* \cdot \mathbf{d}$	0	1/2	−0.0006	0.4954	−0.0001	1.0135	0.0003	1.0149

Table H.1: Means and variances of dot-products for decoding and comparing bindings. The experimental results for the circular system are for $n = 512$ and 10,000 runs. The last four columns of data for the corresponding operations in the normalized standard system are from Table G.1, repeated here for easy comparison. There is only one set of statistics for circular vectors because the decoding/similarity identity holds.

of convolution expressions from Table 3.1. For "hits" in decoding (or comparison), the dot-product is one and the variance is zero, because the decoding is exact. For "misses", the expected dot-product is zero, and the variance is 1/2, as with $\theta \cdot \phi$ in the previous section. The means and variances of dot-products involving higher-order bindings follow the same simple pattern. Results from simulations are shown in Table H.1; these agree with these calculations.

There is no need to consider decoding and similarity separately, as there was with normalized standard vectors, since the decoding/similarity identity

$$(\mathbf{a} \circledast \mathbf{b}) \circledast \mathbf{c}^* \cdot \mathbf{d} = (\mathbf{a} \circledast \mathbf{b}) \cdot (\mathbf{c} \circledast \mathbf{d})$$

holds for circular vectors:

$$\begin{aligned} (\theta \odot \phi) \odot (-\psi) \cdot \gamma &= \cos((\theta + \phi) - \psi - \gamma) \\ &= \cos((\theta + \phi) - (\psi + \gamma)) \\ &= (\theta \odot \phi) \cdot (\psi \odot \gamma) \end{aligned}$$

H.3 Results on analogical similarity estimation

Circular vectors can be used for estimation of analogical similarity. I ran Experiment 3 from Chapter 6 with circular vectors (with 2048 angles). The results are very similar to what is achieved with the standard representation. The means come in the same rank order, but there are more individual violations of the ordering

$$\text{LS} > \text{AN}^{\text{cm}} > (\text{AN}_2, \text{AN}_1) > (\text{SS}^{\times\text{I}}, \text{SS}^{\times\text{H}}, \text{SS}^{-\text{H}}) > (\text{FA}^{\times\text{I}}, \text{FA}^{\times\text{H}}) > \text{OO}_1 > \text{OO}_2$$

P: Spot bit Jane, causing Jane to flee from Spot.

Episodes in long-term memory:		Type	Dot-products Standard Avg	Standard Sd	Circular Avg	Circular Sd
E1:	Fido bit John, causing John to flee from Fido.	LS	0.81	0.011	0.63	0.010
E2:	Fred bit Rover, causing Rover to flee from Fred.	AN^{cm}	0.69	0.016	0.54	0.012
E3:	Felix bit Mort, causing Mort to flee from Felix.	AN_1	0.61	0.018	0.48	0.013
E4:	Mort bit Felix, causing Felix to flee from Mort.	AN_2	0.61	0.018	0.48	0.014
E5:	Rover bit Fred, causing Rover to flee from Fred.	$\text{SS}^{\times\text{I}}$	0.53	0.025	0.40	0.013
E6:	John fled from Fido, causing Fido to bite John.	$\text{SS}^{\times\text{H}}$	0.53	0.023	0.35	0.015
E7:	Mort bit Felix, causing Mort to flee from Felix.	$\text{FA}^{\times\text{I}}$	0.39	0.022	0.31	0.016
E8:	Mort fled from Felix, causing Felix to bite Mort.	$\text{FA}^{\times\text{H}}$	0.39	0.026	0.27	0.015
E9:	Fido bit John, John fled from Fido.	$\text{SS}^{-\text{H}}$	0.51	0.020	0.35	0.012
E10:	Fred stroked Rover, causing Rover to lick Fred.	OO_1	0.25	0.024	0.19	0.017
E11:	Fred stroked Rover, Rover licked Fred.	OO_2	0.12	0.022	0.06	0.017

Table H.2: Results from Analogy Experiment 3 for normalized standard vectors and for circular vectors.

(11 violations for circular vectors compared to 0 for standard vectors). However, these violations are all with the order of SS and FA, which are less important. Overall, the means are somewhat lower, which is not good, but the variances are lower and more uniform, which is good.

Table H.2 shows the means and variances for each of the comparisons, for Experiment 3 and vectors with dimension 2048. The means for standard vectors are the same as in Table 6.7.

Appendix I

Arithmetic tables: an example of HRRs with many items in memory

Distributed representations are generally more efficient than local representations, because they allow many more than n objects to be represented over n units. The analysis of capacity given in Appendix D indicates that HRRs should share this property. In this Appendix I describe a simulation which demonstrates this. I store approximately 5000 different HRRs in a clean-up memory, using 512 dimensional vectors.

Also, this simulation provides a counterexample to Halford *et al*'s [to appear] claim that role-filler representations do not permit one component of a relation can be retrieved given the others.

In Section 3.7 I suggested that a fast estimate of the dot-product (computing using bitwise comparison) could be used to speed up the comparison process. I used this technique in the simulations reported here, and found that it worked well.

I.1 The objects and relations

The base vectors, which have elements distributed as $\mathrm{N}(0, 1/n)$ ($n = 512$) are:

$$\begin{array}{ll} \textbf{optimes} & \textbf{opplus} \\ \textbf{operand} & \textbf{result} \\ \textbf{number}_x \text{ for } & x = 0..2500 \end{array}$$

The relations concern multiplication and addition. They are statements like "two times eight equals sixteen". In general, the relations are:

$$\begin{aligned} \textbf{times}_{x,y} &= \langle \textbf{optimes} + \textbf{operand} * (\textbf{number}_x + \textbf{number}_y) + \textbf{result} \circledast \textbf{number}_{x*y} \rangle \\ \textbf{plus}_{x,y} &= \langle \textbf{opplus} + \textbf{operand} * (\textbf{number}_x + \textbf{number}_y) + \textbf{result} \circledast \textbf{number}_{x+y} \rangle \end{aligned}$$

where x and y range from 0 to 50 with $y \leq x$. I use the same role for each operand, since they have equivalent status ($x \times y = y \times x$).

There are 1326 instances of each relation. Examples are:

$$\begin{aligned} \textbf{times}_{21,12} &= \langle \textbf{optimes} + \textbf{operand} * (\textbf{number}_{21} + \textbf{number}_{12}) + \textbf{result} \circledast \textbf{number}_{252} \rangle \\ \textbf{plus}_{42,25} &= \langle \textbf{opplus} + \textbf{operand} * (\textbf{number}_4 2 + \textbf{number}_2 5) + \textbf{result} \circledast \textbf{number}_{67} \rangle \end{aligned}$$

The 2501 number vectors, along with the 2652 relation vectors, are stored in the clean-up memory.

I.2 Queries to the memory

We can "look up" a relation by supplying sufficient information to distinguish it from other relations. For example, we can look up "21 times 12 equals 252" by finding the most similar relation to any of the following:

$$\begin{array}{l} \langle \textbf{optimes} + \textbf{operand} * (\textbf{number}_{21} + \textbf{number}_{12}) \rangle \\ \langle \textbf{optimes} + \textbf{operand} * \textbf{number}_{12} + \textbf{result} \circledast \textbf{number}_{252} \rangle \\ \langle \textbf{optimes} + \textbf{operand} * \textbf{number}_{21} + \textbf{result} \circledast \textbf{number}_{252} \rangle \\ \langle \textbf{operand} * (\textbf{number}_{21} + \textbf{number}_{12}) + \textbf{result} \circledast \textbf{number}_{252} \rangle \end{array}$$

In the first three cases, the remaining component can be found by decoding the retrieved relation with the role vector for the missing component, e.g.,

$$\textbf{times}_{21,12} \circledast \textbf{result}^*,$$

and retrieving the most similar vector in the clean-up memory, which will be $\textbf{number}_{252}$ for this example. To discover a missing relation name, we need to have a separate clean-up memory containing only relation names (or use an alternative encoding in which there is a role for relation names).

I tried one run of the system, with $n = 512$, making a query for each component missing in every relation. This amounted to 10,608 queries. After retrieving the relation, I decoded the missing component (except for the relation name). This was a further 7,956 queries. The system made no errors.

Relations can be retrieved using fewer components. For example, the most similar relation to $\langle \textbf{result} \circledast \textbf{number}_{221} \rangle$ is $\textbf{times}_{17,13}$.

The above retrieval process requires that we know the roles of each component. If we wish to be able to retrieve relations given some components whose roles we do not know, then we must store the fillers in plain form in the relation, e.g.,

$$\begin{aligned} \textbf{times}_{x,y} = \; & \langle \textbf{optimes} + \langle \textbf{number}_x + \textbf{number}_y + \textbf{number}_{x*y} \rangle \\ & + \textbf{operand} * (\textbf{number}_x + \textbf{number}_y) + \textbf{result} \circledast \textbf{number}_{x*y} \rangle \end{aligned}$$

This is the type of representation I used for the analogy experiments (Chapter 6).

I.3 Using fast estimates of the dot-product

Comparing a 512-dimensional vector against 5,153 vectors in clean-up memory takes considerable time. The technique described in Section 3.7 reduces the computation by using a fast bitwise estimate of the dot-product to judge when it is necessary to compute the floating-point dot-product. Table I.1 shows the CPU times[1] for 10,608 accesses to clean-up memory, with each access involving comparing against 5,153 vectors.

The first column is half the width of the interval s (see Section 3.7 for a definition), in approximate standard deviations. A spread of 4 standard deviations on each side should be very safe. The second column shows the number of times that the use of the fast estimate resulted in the wrong vector being selected by the clean-up memory. The third column

[1] On an SGI machine with a MIPS R4400 CPU and a MIPS R4010 FPU, and a 150MHz clock.

Spread	Errors	Total Time (seconds)	Percent of F.P. dot-products
N/A	0	6824	100
4	0	4287	46
3	0	3512	33
2	0	2876	23
1	0	2002	9.1
0.5	0	1623	3.1
0.1	1	1438	0.60
0	1	1460	0.49

Table I.1: CPU times from runs using fast bitwise comparisons. The times are for 10,608 accesses to a clean-up memory containing 5,135 512-dimensional vectors.

shows the total execution time for the entire program, including the construction of the base and relations vectors and the loading of clean-up memory. This startup time was negligible – around 80 seconds. The last column shows the percentage of vectors for which the floating-point dot-product was calculated. The first row is from a run which did not use the fast estimates at all.

The number of errors is surprisingly low, even for very low spreads. This is probably due to the fact that in this task the best match is nearly always significantly better than the next best match. This makes it very likely that the fast estimate of the dot-product for the true best match is greater than the fast estimate for any other vector. In general, it is probably not safe to use low spreads.

The speedup achieved ($\approx$ 1.4 to 4 times) is reasonable, but not exceptionally good. The speedup will be greater for higher n and for more vectors in the clean-up memory. Further speedup could probably be gained by optimizing the code and the memory organization for the fast comparisons.

References

Abramowitz, M. and Stegun, I. A., editors 1965. *Handbook of mathematical functions with formulas, graphs, and mathematical tables*. Dover, New York.

Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. 1985. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9:147–169.

Ajjanagadde, V. and Shastri, L. 1991. Rules and variables in neural nets. *Neural Computation*, 3:121–134.

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

Anderson, J. A. 1973. A theory for the recognition of items from short memorized lists. *Psychological Review*, 80(6):417–438.

Anderson, J. A., Spoehr, K. T., and Bennet, D. J. 1991. A study in numerical perversity: Teaching arithmetic to a neural network. Technical Report 91-3, Department of Cognitive and Linguistic Science, Brown University.

Borsellino, A. and Poggio, T. 1973. Convolution and correlation algebras. *Kybernetik*, 13:113–122.

Brachman, R. J. 1985a. On the epistemological status of semantic networks. In Brachman, R. J. and Levesque, H., editors, *Readings in Knowledge Representation*, pages 191–216. Morgan Kaufman.

Brachman, R. J. 1985b. Prologue to reflection and semantics in a procedural language. In Brachman, R. J. and Levesque, H., editors, *Readings in Knowledge Representation*, pages 31–40. Morgan Kaufman.

Brigham, E. O. 1974. *The Fast Fourier Transform*. Prentice Hall, Inc., New Jersey.

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

Charniak, E. and Santos, E. 1987. A connectionist context-free parser which is not context-free but then it is not really connectionist either. In *Proceedings of 9th Annual Conference of the Cognitive Science Society*.

Cottrell, G. W. and Small, S. L. 1983. A connectionist scheme for modeling word-sense disambiguation. *Cognition and Brain Theory*, 6:89–120.

Das, S., Giles, C. L., and Sun, G.-Z. 1992. Learning context-free grammars: Capabilities and limitations of a recurrent neural network with an external stack memory. In *Proceedings of the Fourteenth Annual Conference of the The Cognitive Science Society*.

Das, S. and Mozer, M. C. 1994. A hybrid gradient-descent/clustering technique for finite state machine induction. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6 (NIPS*93)*, pages 19–26, San Mateo, CA. Morgan Kaufmann.

Davis, P. J. 1979. *Circulant matrices*. John Wiley & Sons, New York.

Derthick, M. 1990. Mundane reasoning by settling on a plausible model. *Artificial Intelligence*, 46(1-2):107–158.

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

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

Elliot, D. F. 1986. *Handbook of Digital Signal Processing Engineering Applications*. Academic Press, Inc, San Diego, CA.

Elman, J. 1991. Distributed representations, simple recurrent networks and grammatical structure. *Machine Learning*, 7(2/3):195–226.

Elman, J. L. 1988. Finding structure in time. Technical Report CRL-8801, Center for Research in Language, University of California, San Diego.

Elman, J. L. 1990. Finding structure in time. *Cognitive Science*, 14:179–211.

Etherington, D. W. and Reiter, R. 1983. On inheritance hierarchies with exceptions. In *Proceedings AAAI-83*, pages 104–108.

Falkenhainer, B., Forbus, K. D., and Gentner, D. 1989. The Structure-Mapping Engine: Algorithm and examples. *Artificial Intelligence*, 41:1–63.

Feldman, J. A. and Ballard, D. H. 1982. Connectionist models and their properties. *Cognitive Science*, 6:205–254.

Fisher, A. D., Lippincott, W. L., and Lee, J. N. 1987. Optical implementations of associative networks with versatile adaptive learning capabilities. *Applied Optics*, 26(23):5039–5054.

Fodor, J. A. and Pylyshyn, Z. W. 1988. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28:3–71.

Gabel, R. A. and Roberts, R. A. 1973. *Signals and Linear systems*. John Wiley & Sons, Inc.

Gabor, D. 1968a. Holographic model for temporal recall. *Nature*, 217:1288–1289.

Gabor, D. 1968b. Holographic model for temporal recall. *Nature*, 217:1288–1289.

Geman, S. and Geman, D. 1984. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741.

Gentner, D. and Forbus, K. D. 1991. MAC/FAC: A model of similarity-based retrieval. In *Proceedings of the Thirteenth Annual Cognitive Science Society Conference*, pages 504–509, Hillsdale, NJ. Erlbaum.

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

Gentner, D., Rattermann, M. J., and Forbus, K. D. 1993. The roles of similarity in transfer: Separating retrievability from inferential soundness. *Cognitive Psychology*, 25:431–467.

Giles, C. L., Sun, G. Z., Chen, H. H., Lee, Y. C., and Chen, D. 1990. Higher order recurrent networks and grammatical inference. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 2*, pages 380–387, San Mateo, CA. Morgan Kaufmann.

Giles, L., Miller, C., Chen, D., Chen, H., Sun, G., and Lee, Y. 1992. Learning and extracting finite-state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405.

Goldstone, R. L., Medin, D. L., and Gentner, D. 1991. Relational similarity and the nonindependence of features in similarity judgements. *Cognitive Psychology*, 23:222–262.

Hadley, J. A., Healy, A. F., and Murdock, B. B. 1992. Output and retrieval interference in the missing-number task. *Memory and Cognition*, 20:69–82.

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

Harris, C. 1989. Connectionist explorations in cognitive linguistics. Unpublished.

Hinton, G. E. 1981. Implementing semantic networks in parallel hardware. In Hinton, G. E. and Anderson, J. A., editors, *Parallel Models of Associative Memory*. Hillsdale, NJ: Erlbaum.

Hinton, G. E. 1986. Learning distributed representations of concepts. In *Proc. Eighth Annual Conference of the Cognitive Science Society*, pages 1–12. Hillsdale, NJ: Erlbaum.

Hinton, G. E. 1987. Representing part-whole heirarchies in connectionist networks. Unpublished manuscript.

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

Hinton, G. E., McClelland, J. L., and Rumelhart, D. E. 1986. Distributed representations. In D. E. Rumelhart, J. L. M. and the PDP research group, editors, *Parallel distributed processing: Explorations in the microstructure of cognition.*, volume 1, pages 77–109. The MIT Press, Cambridge, MA.

Holyoak, K. J. and Thagard, P. 1989. Analogical mapping by constraint satisfaction. *Cognitive Science*, 13:295–355.

Hopfield, J. J. 1982. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences U.S.A.*, 79:2554–2558.

Hummel, J. E. and Holyoak, K. J. 1992. Indirect analogical mapping. In *Proceedings of the Fourteenth Annual Cognitive Science Society Conference*, pages 516–521.

Humphreys, M. S., Bain, J. D., and Pike, R. 1989. Different ways to cue a coherent memory system: A theory for episodic, semantic, and procedural tasks. *Psychological Review*, 96(2):208–233.

Johnson, N. F. 1972. Organization and the concept of a memory code. In Melton, A. W. and Martin, E., editors, *Coding processes in human memory*, pages 125–159. Winston, Washington DC.

Jordan, M. I. 1986. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Cognitive Science Society Conference*, Hillsdale, NJ. Erlbaum.

Kanerva, P. 1988. *Sparse Distributed Memory*. MIT Press, Cambridge, MA.

Le Cun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. 1990. Handwritten digit recognition with a back-propagation network. In Touretzky, D., editor, *Advances in Neural Information Processing Systems*, volume 2, pages 396–404, Denver. Morgan Kaufmann, San Mateo.

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

Levesque, H. J. and Brachman, R. J. 1985. A fundamental tradeoff in knowledge representation and reasoning (revised version). In Brachman, R. J. and Levesque, H., editors, *Readings in Knowledge Representation*, pages 41–70. Morgan Kaufman.

Lewandowsky, S. and Murdock, B. B. 1989. Memory for serial order. *Psychological Review*, 96(1):25–57.

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

Longuet-Higgins, H. C. 1968. Holographic model of temporal recall. *Nature*, 217:104.

MacLennan, B. 1991. Characteristics of connectionist knowledge representation. Technical Report CS-91-147, Computer Science Department, University of Tennessee, Knoxville.

Markman, A. B., Gentner, D., and Wisniewski, E. J. 1993. Comparison and cognition: Implications of structure-sensitive processing for connectionist models. Unpublished manuscript.

Marr, D. 1982. *Vision*. Freeman, San Francisco.

Maskara, A. and Noetzel, A. 1992. Forcing simple recurrent neural networks to encode context. In *Proceedings of the 1992 Long Island Conference on Artificial Intelligence and Computer Graphics*.

McClelland, J., Rumelhart, D., and the PDP Research Group 1986. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. The MIT Press, Cambridge.

McClelland, J. L. and Kawamoto, A. H. 1986. Mechanisms of sentence processing: Assigning roles to constituents. In J. L. McClelland, D. E. R. and the PDP research group, editors, *Parallel distributed processing: Explorations in the microstructure of cognition*, volume 2, pages 272–326. The MIT Press, Cambridge, MA.

Metcalfe, J. 1991. Recognition failure and CHARM. *Psychological Review*, 98(4):529–553.

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

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

Miikkulainen, R. 1993. *Subsymbolic Natural Language Processing*. The MIT Press, Cambridge, MA.

Miikkulainen, R. and Dyer, M. G. 1989. Encoding input/output representations in connectionist cognitive systems. In Touretzky, D. S., Hinton, G. E., and Sejnowski, T. J., editors, *Proceedings of the 1988 Connectionist Models Summer School*, San Mateo, CA. Morgan Kaufman.

Miller, G. A. 1956. The magical number seven plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63:81–97.

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

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

Murdock, B. B. 1985. Convolution and matrix systems: A reply to pike. *Psychological Review*, 92(1):130–132.

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

Murdock, B. B. 1992. Serial organization in a distributed memory model. In Healy, A., Kosslyn, S., and Shiffrin, R., editors, *From learning theory to connectionist theory: Essays in hoor of William K. Estes*, volume 1, pages 201–225. Erlbaum, Hillsdale, NJ.

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

Neal, R. M. 1992. Connectionist learning of belief networks. *Artificial Intelligence*, 56:71–113.

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

Pearl, J. 1988. *Probabilisitic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA.

Pike, R. 1984. Comparison of convolution and matrix distributed memory systems for associative recall and recognition. *Psychological Review*, 91(3):281–294.

Pinker, S. and Prince, A. 1987. On language and connectionism: Analysis of a parallel distributed processing model of language acquisition. Technical Report Occasional Paper 33, Department of Brain and Cognitive Science, MIT, Cambridge, MA 02137. Also published in Cognition, Volume 28, Numbers 1-2, March 1988.

Plate, T. A. in press. Holographic reduced representations. *IEEE Transactions on Neural Networks*. To appear.

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

Reichardt, W. 1957. Autokorrelations-auswertung als funktionsprinzip des zentralnewvensystems (Auto-correlations as a functional principle of the central nervous system). *Z. Naturforsch*, 12b:448–457. Cited in Borsellino and Poggio [1973] and Schönemann [1987].

Rosenfeld, R. and Touretzky, D. S. 1987. Four capacity models for coarse-coded symbol memories. Technical Report CMU-CS-87-182, Carnegie-Mellon University.

Rosenfeld, R. and Touretzky, D. S. 1988. Coarse-coded symbol memories and their properties. *Complex Systems*, 2(4):463–484.

Ross, B. 1989. Distinguishing types of superficial similarities: Different effects on the access and use of earlier problems. *Journal of Experimental Psychology Learning Memory Cognition*, 15:456–468.

Rumelhart, D. E., Hinton, G. E., and J., W. R. 1986. Learning internal representations by error propagation. In *Parallel distributed processing: Explorations in the microstructure of cognition*, volume 1, pages 318–362. The MIT Press, Cambridge, MA.

Rumelhart, D. E. and McClelland, J. L. 1986a. On learning the past tenses of english verbs. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2, pages 216–271. The MIT Press, Cambridge, MA.

Rumelhart, D. E. and McClelland, J. L. 1986b. Pdp models and general issues in cognitive science. In D. E. Rumelhart, J. L. M. and the PDP research group, editors, *Parallel distributed processing: Explorations in the microstructure of cognition*, volume 1, pages 110–146. The MIT Press, Cambridge, MA.

Schönemann, P. H. 1987. Some algebraic relations between involutions, convolutions, and correlations, with applications to holographic memories. *Biological Cybernetics*, 56:367–374.

Seifert, C. M. and Hammond, K. J. 1989. Why there's no analogical transfer. In *Proceedings: Case-Based Reasoning Workshop*, pages 148–152. Morgan Kaufmann.

Sejnowski, T. J. and Rosenberg, C. R. 1986. *NETtalk: A parallel network that learns to read aloud*. Technical report 86-01, Department of Electrical Engineering and Computer Science, Johns Hopkins University, Baltimore, MD.

Selman, B. and Hirst, G. 1985. A rule-based connectionist parsing system. In *Proceedings of the Seventh Annual conference of the Cognitive Science Society*, pages 212–221, Hillsdale N.J. Erlbaum.

Servan-Schreiber, D., Cleeremans, A., and McClelland, J. L. 1991. Graded state machines: The representation of temporal contingencies in simple recurrent networks. *Machine Learning*, 7(2/3):161–194.

Shastri, L. 1988. *Semantic networks: an evidential formalization and its connectionist realization*. Pitman, London.

Simard, P. and LeCun, Y. 1992. Reverse TDNN: an architecture for trajectory generation. In Moody, J. M., Hanson, S. J., and Lippman, R. P., editors, *Advances in Neural Information Processing Systems 4 (NIPS*91)*, Denver, CO. Morgan Kaufmann.

Slack, J. N. 1984a. A parsing architecture based on distributed memory machines. In *Proceedings of COLING-84*, pages 92–95, Stanford, Calif. Association for Computational Linguistics.

Slack, J. N. 1984b. The role of distributed memory in natural language processing. In O'Shea, T., editor, *Advances in Artificial Intelligence: Proceedings of the Sixth European Conference on Artificial Intelligence, ECAI-84*. Elsevier Science Publishers.

Slack, J. N. 1986. A parsing architecture based on distributed memory machines. In *Proceedings of COLING-86*, pages 476–481. Association for Computational Linguistics.

Smolensky, P. 1986. Neural and conceptual interpretation of PDP models. In D. E. Rumelhart, J. L. M. and the PDP research group, editors, *Parallel distributed processing: Explorations in the microstructure of cognition*, volume 2, pages 390–431. The MIT Press, Cambridge, MA.

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

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

Sopena, J. M. 1991. ESRP: A distributed connectionist parser that uses embedded sequences to represent structure. Distributed on the *connectionists* mailing list.

St. John, M. F. and McClelland, J. L. 1990. Learning and applying contextual constraints in sentence comprehension. *Artificial Intelligence*, 46:217–257.

Thagard, P., Holyoak, K. J., Nelson, G., and Gochfeld, D. 1990. Analog Retrieval by Constraint Satisfaction. *Artificial Intelligence*, 46:259–310.

Touretzky, D. S. 1986a. Boltzcons: Reconciling connectionism with the recursive nature of stacks and trees. In *Proceedings of the Eighth Annual conference of the Cognitive Science Society*, Hillsdale N.J. Erlbaum.

Touretzky, D. S. 1986b. *The Mathematics of Inheritance Systems*. Pitman, London.

Touretzky, D. S. 1986c. Representing and transforming recursive objects in a neural network, or "Trees *do* grow on Boltzmann machines". In *Proceedings of the 1986 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE. Atlanta, GA.

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

Touretzky, D. S. and Geva, S. 1987. A distributed connectionist representation for concept structures. In *Proceedings of the Ninth Annual Cognitive Science Society Conference*, Hillsdale, NJ. Erlbaum.

Touretzky, D. S. and Hinton, G. E. 1985. Symbols among the neurons: Details of a connectionist inference architecture. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 238–243, San Mateo, CA. Morgan Kaufmann.

Touretzky, D. S. and Hinton, G. E. 1988. A distributed connectionist production system. *Cognitive Science*, 12(3):423–466.

Tversky, A. 1977. Features of similarity. *Psychological Review*, 84(4):327–352.

van Gelder, T. 1990. What is the "D" in "PDP"? An overview of the concept of distribution. In Stich, S., Rumelhart, D., and Ramsey, W., editors, *Philosophy and Connectionist Theory*. Erlbaum, Hillsdale N.J.

Waltz, D. L. and Pollack, J. B. 1985. Massively parallel parsing: A strongly interactive model of natural language interpretation. *Cognitive Science*, 9:51–74.

Watrous, R. and Kuhn, G. 1992. Induction of finite state languages using second-order recurrent networks. *Neural Computation*, 4(3):406–414.

Weber, E. U. 1988. Expectation and variance of item resemblance distributions in a convolution-correlation model of distributed memory. *Journal of Mathematical Psychology*, 32:1–43.

Wharton, C. M., Holyoak, K. J., Downing, P. E., Lange, T. E., Wickens, T. D., and Melz, E. R. to appear. Below the surface: Analogical similarity and retrieval competition in reminding. *Cognitive Psychology*.

Williams, R. J. and Zipser, D. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280.

Willshaw, D. 1981a. Comments on chapter 3 (Holography, associative memory, and inductive generalization). In Hinton, G. E. and Anderson, J. A., editors, *Parallel models of associative memory*, pages 99–101. Erlbaum, Hillsdale, NJ, updated edition.

Willshaw, D. 1981b. Holography, associative memory, and inductive generalization. In Hinton, G. E. and Anderson, J. A., editors, *Parallel models of associative memory*, pages 103–124. Erlbaum, Hillsdale, NJ, updated edition.

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

Zemel, R. S. and Hinton, G. 1994. Developing population codes by minimizing description length. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6 (NIPS*93)*, pages 11–18, San Mateo, CA. Morgan Kaufmann.